Programming the Mark I: Early Programming Activity at the University of Manchester

MARTIN CAMPBELL-KELLY

Computer activity at Manchester University began in 1946 with the construction of a CRT-based memory, which was followed by a series of prototype computers. The work culminated in the Ferranti Mark I, completed in early 1951. This paper describes the programming systems devised, first for the prototype and then for the production Mark I, and includes an account of two novel automatic coding schemes developed during 1952 and 1954. The paper concludes with an assessment of the programming activity.

Keywords and phrases: subroutine, loader, assembler, compiler, interpreter, paging, operating system

CR categories: 1.2, 2.43, 4.11, 4.12, 4.13, 4.21, 4.22, 4.31, 4.35, 4.42

1 Introduction

By 1950, there were three influential centers of programming in Britain where working computers had been constructed: Cambridge University (the EDSAC), Manchester University (the Mark I), and the National Physical Laboratory at Teddington (the Pilot ACE). At each of these centers, distinctive styles of programming evolved, largely independently of each other. Through the activities of these centers and their commercially produced derivative machines (the LEO, the Ferranti Mark I, and the DEUCE, respectively), the foundations of programming were laid in Britain during the early 1950s. Other influences were at work, of course, such as A. D. Booth's work at Birkbeck College of London University that led to the successful HEC series of computers (Booth 1975); there were also numerous contemporary developments in the United States. But none of these had nearly as much influence on programming in Britain as the three leading British centers at Cambridge, Manchester, and Teddington, where many early British programmers were initiated by means of programming schools, research studentships, secondments, and industrial collaboration.

This paper looks in detail at the early development of programming at Manchester University. Early programming activity at Cambridge University was described in a companion paper (Campbell-Kelly, *Annals of the History of Computing*, Vol. 2, No. 1, January 1980), and programming at the National Physical Laboratory is the subject of a third paper, in preparation.

Early programming at Manchester centered on the Mark I computers, which was the name ascribed both to the prototype machines built at the university and to the commercially produced Ferranti Mark I, which spanned an era from 1951 until 1958. Key dates include the following:

June 1948: "baby machine" first worked

April 1949 to about August 1950: Mark I prototypes in use

February 1951: Ferranti Mark I installed at Manchester University

December 1958: maintenance discontinued on Ferranti Mark I

The first part of this paper describes the early programming systems developed at Manchester; Section 2 describes the programming schemes devised by A. M. Turing for the prototype Mark I (in late 1949) and the production Mark I (in early 1951); Section 3 illustrates the programming system for the engineered Mark I by outlining the preparation of a complete programming example. Partly because of the inadequacies of the programming system, very early work on automatic coding was done during 1952–1955 at Manchester, and Section 4 describes this. Finally, Section 5 gives an assessment of the whole programming activity.

Terminology

Perhaps a note on terminology is appropriate at this point. In general, contemporary terminology will be used, partly to evoke the period described, and partly to avoid the risk of creating a false impression by the use of a modern term. (For example, the old term *input routine* will be used rather than *loader* or *assembler*; indeed, neither of these would be quite appropriate because an input routine fell somewhere between the two.) I have also elected to retain the early British spelling *programme* and the hyphen in *sub-routine*, again to be in keeping with the period (and with perhaps just a little chauvinism).

Author's address: Department of Mathematics and Computer Studies, Sunderland Polytechnic, Sunderland SR1 3SD, England.

©1980 AFIPS 0164–1239/80/020130–168$01.00/0

1.1 Background[1]

In July 1946, M. H. A. Newman, F.R.S., recently appointed Fielden Professor of Pure Mathematics at the University of Manchester, had obtained a Royal Society grant for a projected Computing Machine Laboratory. Newman was a member of the selection committee that appointed F. C. Williams to the Chair of Electro-technics; at the time of his appointment Williams was actively working on a novel digital store using cathode ray tubes.

Williams continued this work when he took office at the University in December 1946, and he brought in an assistant, T. Kilburn. During 1947 the Williams tube store, as it came to be known, was developed to a point where it could satisfactorily store 2048 bits on a single CRT. The store was incorporated in a minimal computer in order to establish its viability: the "baby machine" began operation in June 1948 and was the first electronic stored-program computer to be completed. With only 32 words of store, however, it was not capable of solving realistic problems.

The baby machine was extended by a series of modifications to a state where it was capable of doing useful work by April 1949, when it had become equipped with 128 words of Williams tube storage supplemented by a magnetic drum. By October 1949, the machine had received further improvements, including paper-tape input and output, and was used on a fairly regular basis for the solution of problems in various fields. This machine was shut down in August 1950.

In November 1948, a contract had been placed with Ferranti Limited, a Manchester-based manufacturer of electrical and electronic equipment, for the construction of the Mark I. The logic design was completed late in 1949 and the machine was available from February 1951. Incidentally, there was some confusion over the name of the production Mark I in the early literature, where it was variously called MADM, Manchester Electronic Computer Mark II, Ferranti Mark I, M.U.D.C., and M.U.E.D.C. In this paper, the computer is called the "Mark I," which was what contemporary users called it. (The prototype machine was also called the Mark I, of course, but it should be clear in this account which machine is being referred to; the qualifiers "production" or "prototype" will be used where necessary.)

After an inaugural conference for the production Mark I in July 1951, the Computing Machine Laboratory became a thriving center of computing in the north of England and began to attract many regular users from both within the university and outside. During 1952, the laboratory acquired a small but very competent staff who provided consultation, programming advice, and instruction. The impact of the laboratory within the university was considerable and it attracted much interest from the science and engineering departments. By 1953, some 50 people had been trained to use the machine. Outside users included representatives from industry, government establishments, and research organizations; the programming group within Ferranti had particularly close links with the laboratory. Many of these early users are now significant names in the British computer community.

Ferranti sold a second Mark I to the University of Toronto and produced a follow-up machine with a slightly rationalized order code, the Mark I*, of which seven were sold. The collaboration between the university and Ferranti was exceptionally fruitful; Ferranti marketed two further machines, the Mercury and Atlas, based on Manchester University designs, before disposing of their computer interests in 1963. A Mercury computer was installed in the laboratory in October 1957, and over the next year the Mark I gradually went out of use; it was dismantled in June 1959.

Manchester University has long been recognized as one of the world's leading centers of hardware innovation, arguably reaching a peak with the virtual memory of the Atlas computer. There have also been some notable software successes, especially the compiler-compiler (Brooker et al. 1964) and the Atlas Supervisor (Kilburn et al. 1961). Manchester has also played a leading role in computer education, having instituted the first university undergraduate course in computer science in Britain in 1965. Through its graduate and postgraduate students, as well as its wealth of technical innovation and industrial collaboration, Manchester University continues to play a major role in the British computer scene.

1.2 People

The two principal people in early computer development at Manchester University were M. H. A. Newman and F. C. Williams. Newman had a considerable reputation as a pure mathematician, gained before the war, and he had spent the war years engaged on cryptography at Bletch-

[1] Background material has been obtained from a numher of publications on the history of Manchester computers: Lavington (1975), *Computer Weekly* (1969), Williams (1975).

ley Park where he had been associated with the development of special-purpose electronic code-breaking computers (Randell 1976). His influence on the Mark I project was very important: in obtaining Royal Society funds, in his enthusiasm for and encouragement of the project, and in helping to specify the functional requirements of the machine.

Williams had a background as an extraordinarily inventive electrical engineer (Kilburn and Piggot 1978). His war years were spent on radar work at the Telecommunications Research Establishment (TRE). CRT storage was originally conceived in connection with storing radar data for echo cancellation. Developing this basic concept into a workable digital store was an outstanding engineering achievement. Williams's work on computers was a relatively short span of a long career as an electrical engineer, and his finest achievements lie in that field. He was the recipient of many academic honors, was elected to the Royal Society in 1950, and was knighted in 1976, a year before his death.

Martin Campbell-Kelly is Senior Lecturer in the Department of Mathematics and Computer Studies, Sunderland Polytechnic, England. He is a graduate of the Department of Computer Science, University of Manchester.

Williams brought in T. Kilburn to assist him in the development of the CRT store. Kilburn had graduated in mathematics at Cambridge University in 1942 and spent the remaining war years working under Williams on radar and other developments at TRE. In 1952, Kilburn assumed full responsibility for computer developments at Manchester University, when Williams's interests turned to other things. Kilburn's involvement with Mark I was mainly on engineering and logic design; in this he was ably assisted by D. G. B. Edwards, G. C. Tootill, A. A. Robinson, and G. E. Thomas, all now leading British computer scientists. Kilburn's contributions to computer engineering and design have been recognized by numerous academic honors; he was elected to the Royal Society in 1965.

In September 1948, A. M. Turing was appointed Reader in the Department of Mathematics, under Newman, with "the expectation that he would lead the mathematical side of the work" (Newman 1955). Newman and Turing were both closely involved in specifying the requirements of the prototype and production machines and in running very early programmes in number theory. Subsequently, Turing was responsible for the early programming systems. Turing's contributions to computing are well known: the classic "Turing machine" work before the war, work on code-breaking machinery during the war, and the logic design of the ACE computer in 1945 (Carpenter and Doran 1977). Sadly, Turing's programming developments for the Mark I did not match these early achievements and were not an example of his finest work. Turing's interest in computer development declined during 1951 as his interests turned toward morphogenesis, although he remained a heavy user of machine time. He was elected to the Royal Society in 1951.

In October 1949, Cicely Popplewell joined the Computing Machine Laboratory as Turing's programming assistant. Popplewell had a background of computing using a punched-card installation, and the terms of her appointment were that she would assist in the preparation of a library of sub-routines for the prototype Mark I. Audrey Bates, an M.Sc. research student supervised by Turing, arrived a little later. The only other regular user of the prototype Mark I was D. G. Prinz of Ferranti, who used the machine for engineering calculations.

In mid-1951, with the installation of the production Mark I, the Computing Machine Laboratory began to attract many outside users. The first of these were A. E. Glennie and K. N. Dodd of the Armaments Research Establishment. They had both formerly worked on EDSAC and brought with them something of the Cambridge approach to programming. Glennie spent about 60% of his time at Manchester until the end of 1954 and made very interesting contributions to programming system development, in particular an early compiler.

R. A. Brooker, who had also worked with EDSAC, was appointed to the staff of the laboratory in October 1951 to strengthen the programming effort. Brooker, assisted by Popplewell, was responsible for programming system development from this time, assuming the duties that no longer interested Turing. Brooker's most notable achievement was a highly successful automatic programming system, the Mark I Autocode. Brooker remained at Manchester for many years, where his considerable achievements included the Mercury Autocode (Brooker 1958) and the compiler-compiler. Since 1967, Brooker has held the Chair of Computer Science at Essex University.

One visitor to the laboratory of particular interest was C. Strachey. Both Brooker and Popplewell relate an anecdote about their first meeting with him. Strachey, then a master at Harrow School, had obtained a copy of the Mark I programmer's handbook. He wrote a programme to play draughts (checkers) and asked if he could visit the laboratory the following half-term holiday to run it. (This would be October or November 1951.) Permission was given and Strachey sent his programme for punching beforehand. The programme was about 20 pages long (over a thousand instructions) and the naiveté of a first-time user attempting a programme of such a length caused not a little amusement among the programmers in the laboratory. Anyway, the day came and Strachey loaded his programme into the Mark I. After a couple of errors were fixed, the programme ran straight through and finished by playing "God Save the King" on the "hooter" (loudspeaker). On that day Strachey acquired a formidable reputation as a programmer that he never lost. Soon after, Strachey was hired as a consultant by the National Research and Development Corporation at a salary that Harrow School could not match. One of his first jobs was as a consultant on calculations for the St. Lawrence seaway project using the Ferranti Mark I at the University of Toronto.

Two early users of the Mark I, starting in mid-1951, were N. E. Hoskin, assistant lecturer, and R. K. Livesley, research student, in the Department of Mathematics. In August 1952 they took the first two research assistantships in programming in the laboratory. Working under Brooker, they advised outside users and assisted in the development of the sub-routine library. Other contributors to the programming activity included D. C. Gilles, B. Richards, F. H. Sumner, and D. Morris, all currently professors of computer science in Britain—the latter two in the present Department of Computer Science at Manchester.

1.3 Sources

In addition to the open literature (see references) a number of less accessible sources have been used in preparing this paper.

There is very little programming documentation for the prototype Mark I. G. C. Tootill kept a notebook (1948–1949) during his period at the university; it contains many fascinating notes on the development of the prototype machine, but few on programming. The papers of D. G. Prinz, who was actively involved in programming, contain some interesting fragments but little of substance. Audrey Bates's thesis (1950) contains listings of her programmes for symbolic logic, but there is little discussion of the programming system as such. Probably the fullest account of programming for the prototype machine is contained in an appendix to the first programmer's handbook for the production Mark I (HB1). Cicely Popplewell wrote an account of her programming experience with the prototype Mark I in 1969; the material is anecdotal rather than technical (Popplewell 1969*a,b*).

A beautifully printed proceedings of the Inaugural Conference of the production Mark I in July 1951 contains a number of interesting papers and discussions on programming (Manchester 1951). (Incidentally, the proceedings contains a list of the 169 conference delegates; this includes just about everybody in Britain who had an interest in computers at the time.) The various programming systems for the Mark I were well documented in a series of programmer's handbooks issued from 1951 to 1956. The first handbook was written by Turing (HB1); as well as an appreciable number of errors, it contains some fine examples of his delightful prose, of which the following may serve as a specimen: "The engineers ... will consider the removal of a valve or the connection of two points temporarily with crocodile clips to be admissible but would frown on certain users of a hatchet."

Turing's manual was not a model of clarity, and Prinz wrote a simplified handbook for internal use by Ferranti's programming group (Prinz 1952). Brooker produced a revised handbook in August 1952 (HB2) that was duller but more reliable than Turing's. This was followed by a supplement of examples (HB2S) a few months later. A third handbook (HB3) was issued in September 1953 and a supplement to this (HB3S) in January 1956. Most of the handbooks are at present in private hands. In addition to the handbooks, there was a fair amount of documentation of programmes and sub-routine specifications on loose sheets; much of this has been lost, although there is a good deal of early material in the papers of the late C. Strachey.[2]

The Strachey papers contain many items relating to the Mark I, such as the original draughts programme (all written out on the "prep sheets" of Harrow scholars!). Strachey used the CRT monitor tube to display the draughts board, and

[2] Now being cataloged by the Contemporary Scientific Archives Centre, Oxford.

there are photographs of this. There is also a programme for writing love letters and a good deal of material on Strachey's consultancy work on the second Ferranti Mark 1, the FERUT, at the University of Toronto.

Glennie and Brooker have both retained working papers for their autocodes, and I drew on these extensively for Section 4 of this paper. Conversations with Kilburn, Brooker, Glennie, Popplewell, and Prinz have added substantially to the content of this report. Finally, several early users have provided me with written answers to questions.

2 Early Programming System Development

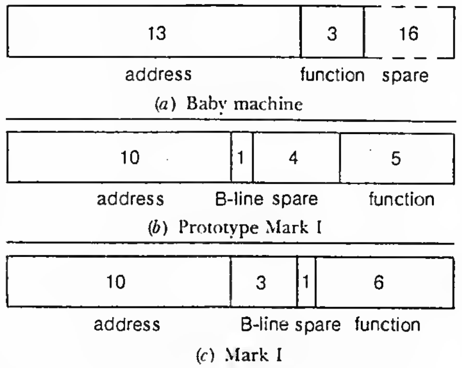
This section describes the programming systems devised by Turing for the prototype Mark I during 1949 and for the production Mark I in 1951. Two further but essentially similar programming schemes devised during 1952–1953 are also described. All the programming systems were based on a simple input routine and a sub-routine library. Before going into programming details, we will discuss the Mark I and its instruction code.

2.1 The Mark I

The history of the Mark I is complicated by the fact that there was an evolutionary process from the first experimental machine through to the production Mark I. To some extent, the evolutionary process was continuous, but from the programmer's point of view four distinct machines can be identified: the baby machine, two prototype versions of the Mark I (called the "improved machine" and the "large-scale machine" by Williams and Kilburn at the Inaugural Conference in July 1951, although in fact most of the hardware was common and there was little difference in size), and the production Mark I. A summary of this evolution is:

June 1948: baby machine working
April 1949: improved machine available
October 1949: large-scale machine available
February 1951: production Mark 1 available

2.1.1 The baby machine. A small-scale experimental computer known as the miniature or baby machine first operated successfully on June 21, 1948, and was the first EDVAC-type electronic stored-program computer to be completed. The logic design was by Kilburn (mainly) and Williams; the object of building the machine was to demonstrate the suitability of Williams tube storage for digital computers.



(*a*) Baby machine

(*b*) Prototype Mark I

(*c*) Mark I

Figure I. Evolution of the instruction format.

The miniature machine is described in considerable detail in Williams et al. (1951). It had 32 words of Williams tube storage and an order code of seven instructions. Figure 1*a* shows the instruction format. Binary programmes were put, bit by bit, into the store using manual keys and a "typewriter" of 32 pushbuttons, each button corresponding to one bit of the store line. "Output" was obtained by inspection of the Williams tube monitor. Williams (1975, p. 330) evocatively described the operation of the first programme to be run:

> When first built, a program was laboriously inserted and the start switch pressed. Immediately the spots on the display tube entered a mad dance. In early trials it was a dance of death leading to no useful result, and what was even worse, without yielding any clue as to what was wrong. But one day it stopped and there, shining brightly in the expected place, was the expected answer. It was a moment to remember.

The programme referred to was a 17-instruction routine to compute the highest factor of an integer.[3] Subsequently, this programme ran for 52 minutes, doing over 3.5 million operations, to compute the highest factor of 2^{18}; the duration of this run amply confirmed the suitability of Williams tube storage. Now the emphasis shifted toward constructing a more ambitious machine.

[3] The original programme has been lost, but there is a later version (July 1948) in Tootill's notebook and another (November 1948) in Prinz's papers.

2.1.2 The improved machine. Several enhancements were made to the baby machine, and the improved machine was doing useful work from April 1949. The word length was increased from 32 to 40 bits to reduce rounding errors. This turned out to be a fortunate choice for it made double-length working at Manchester largely unnecessary. The 40-bit word could also accommodate two 20-bit instructions, as in Figure 1*b*. The CRT storage capacity was increased to 128 words, supplemented by a 1024-word magnetic drum. At this stage transfers between the magnetic and electrostatic store could only be achieved with manual intervention to set appropriate switches.

The order code now consisted of 26 instructions, including orders for setting the B-lines. In retrospect, the most enduring contribution of the Mark 1 was the concept of B-lines, which are now embodied as index registers in all computers. The reason for the term *B-line* was that A and C had already been used for the accumulator and control; presumably the multiplicand register became known as the D-register for a similar reason.

The B-line patent is in the names of Newman, Williams, Kilburn, and Tootill. It was probably a collective rather than an individual invention, although the memories of the participants are not in complete accord as to the actual events. The key features offered by the B-line were:

1. The ability to add a preset word to an instruction before obeying it (i.e., indexed addressing).
2. The provision of extra central registers in addition to the accumulator.
3. The ability to relocate a sub-routine (i.e., use as a base register).

In practice, the third feature—use as a relocation register—was not particularly effective and was never exploited.

Input and output for the improved machine used the same arrangements as the baby machine: a panel of pushbuttons for input and output by inspection of the Williams tube monitor. Because of the primitive input/output, there was no programming system or sub-routine library, of course; programmes were coded in binary and inserted one digit at a time. Even so, useful work was done on the investigation of Mersenne primes in June 1949 and was announced by Newman in a letter to the *Times* (1949*a*) and reported at the Cambridge University Computer Conference the same month (Newman 1949*b*). Kilburn and Tootill wrote the original programme, which was subsequently improved by Turing (Lavington 1975, p. 12).

2.1.3 The large-scale machine. During the summer of 1949 work began on the extension of the improved machine and was completed by the following October. The main enhancements consisted of instructions for programmed drum transfers and the provision of input/output using standard teleprinter equipment. The teleprinter equipment was acquired by Turing through contacts with the Foreign Office Department of Communications at Bletchley Park and installed in June or July. The equipment include a keyboard perforator of German origin ("liberated" during the war) with a "considerable tendency to replace the digit 1 by 0" (HB1, p. 92).

At this time the electrostatic storage was extremely sensitive to electric interference of any kind and for this reason, as well as lack of space, the magnetic drum and teleprinter equipment were housed in a room on the floor above. The terms *up* and *down transfers* persisted for years afterward.

With the addition of the teleprinter input/output equipment, Turing was able to develop a programming system that superseded the former method of entering programmes in binary by means of the panel of pushbuttons. When Popplewell arrived in October 1949 the programming system had just come into use; some years later she wrote her recollections of programming the prototype machine (1969*a*):

> Operating the machine required considerable physical stamina. Starting in the machine room one alerted the engineer and then used the hand switches to bring down and enter the input program. A bright band on the monitor tube indicated that the waiting loop had been entered. When this had been achieved, one ran upstairs and put the tape in the tape reader and then returned to the machine room. If the machine was still obeying the input loop one called to the engineer to switch on the writing current, and cleared the accumulator (allowing the control to emerge from the loop). With luck, the tape was read. As soon as the pattern on the monitor showed that input was ended the engineer switched off the write current to the drum. Programs which wrote to the drum during the execution phase were considered very daring. As every vehicle that drove past was a potential source of spurious digits, it usually took many attempts to get a tape in—each attempt needing another trip up to the tape room.

A small library of sub-routines was developed for the prototype and a small number of full-scale problems tackled. The latter included work by Turing on Mersenne primes and the Riemann hypothesis; Popplewell did work on ray tracing, Bates did work on symbolic logic for her Master's thesis, and Prinz did work on Laguerre functions for an engineering application at Ferranti.[4] The prototype was closed down about August 1950, pending the arrival of the production model.

2.1.4 The Ferranti Mark I. During the summer of 1949, work on the production Mark I had started at Ferranti. Working to the requirements of Newman and Turing, Tootill developed the logic design at Ferranti from August to November 1949. The computer was installed at Manchester University for final testing in February 1951 and was used for programming systems and other development work until it was officially inaugurated in July 1951.

The Mark I, although a fairly conventional single-address machine, had many interesting features of which only a few can be described here. The instruction format, given in Figure 1*c*, was essentially the same as on the prototype, with the operation code field extended to six bits and three bits now used to specify one of eight B-lines. The ten-bit address accommodated an address space of 1024 lines of electrostatic storage, although only 512 lines were actually provided. Incidentally, *line* is almost always used in preference to *word* in the Mark I literature; a long line was 40 bits, a short line 20 bits, and a line usually meant a short line. Of course, a line had a physical analog in the form of the trace on the Williams tube.

Notice that the instruction is in the reverse direction to how we conventionally write instruction formats—the least significant digit of the address is on the left. The same was true for numbers, which were written with the least significant digit on the left and the most significant on the right. The reason for this is that in a serial machine, the digits are produced least-significant digit first; Williams tubes and oscilloscopes conventionally sweep with time going from left to right, so it was natural to write binary numbers that way and this was common on early computers.

The instruction repertoire had been extended to 50 orders and the additional orders included instructions for B-line arithmetic, arithmetic instructions for both signed and unsigned fractions, and a number of miscellaneous instructions. Instruction times were on the order of 1 ms.

Instruction modification in the Mark I is particularly interesting. Unlike present-day machines, the B-line of 20 bits modified the entire instruction and not just the address field. This could result in a change of operation code and/or B-line when the instruction was modified. Some very curious tricks could be, and were, programmed by this means.

Although only 50 of the 64 possible operation orders were defined, decoder minimization was used so that in fact every order did do something. The convention that programmers would not use undefined orders was generally followed.

Transfer-of-control instructions were interesting from a systems programmer's point of view. The address part of a branch instruction was not the destination of the branch, but instead the address of a line containing that destination. This made things such as jump tables very easy to implement and the forward-reference problem nonexistent. There were both absolute and relative versions of branch instructions; the latter were particularly useful for writing position-independent code.

There were no shift orders, shifts being achieved by multiplication by powers of two. The high-speed multiplier was much faster than, and eliminated the need for, a shift unit although it did mean that about 40 lines of store were permanently occupied by powers of two. (Limited shifting could, however, be achieved by an order that doubled the contents of the accumulator.)

At the request of Turing an instruction to generate a random number from a noise source was provided. Unfortunately, the numbers turned out to be not particularly random, and debugging was difficult because programmes were not repeatable; consequently, the instruction eventually fell into disuse. There is a story that a gambling programme counted the number of times the digits 0, 1, . . . , 9 were produced by the random-number generator. F. C. Williams would encourage unsuspecting visitors to play this game (Livesley 1979); he always adjusted the generator beforehand to produce more 1s than 0s and then put his money on 7!

Other curious instructions included a sideways add that gave the number of binary 1s in a word (requested by Newman for problems in number theory) and a "hoot" instruction that pulsed a loudspeaker. Another instruction gave the position of the most significant 1 in a word; this was useful for normalizing floating-point numbers. Abbreviated instruction codes for the Mark I are given in Table 1. This table is not intended to be fully self-explanatory, but it should enable the programmes given in this paper to be understood. (The peculiar operation codes //, /E, etc., will be explained later.)

Peripherals for the Mark I were a high-speed paper-tape reader for input (about 150 characters per second) and a teleprinter and paper-tape punch for output (6⅔ and 15 characters per second, respectively).

[4] The output from Prinz's programme is still in existence, dated August 1950.

Table 1. Mark I instruction set

//	Obey H as magnetic instruction	T/	$L' = S_+, \quad M' = 0$	
/E	$S' = M$	TE	$L' = S' = S \vee L$	
/@	Normalize	T@	L' = 65th line	
/A	$S' = M, \quad A' = L_+$	TA	$S' = L, \quad A' = 0$	
/:	Obey S as magnetic instruction	T:	$A' = 0$	
/S	$S' = L$	TS	$S' = L' = S \vee L, \quad A = 0$	
/I	$L' = M, \quad M' = L$	TI	$A' = A + S_+$	
/U	$S' = L, \quad L' = M, \quad M' = 0$	TU		
/½	$A' = A - DS_+$	T½	$A' = S_\pm$	
/D	$A' = A - DS_\pm$	TD	$A' = A \vee S_\pm$	(logical OR)
/R	Sideways add	TR	$A' = A \,\&\, S_\pm$	(logical AND.)
/J	$M' = M + S$	TJ	$A' = A \neq S_\pm$	(logical EXCLUSIVE OR)
/N	$A' = A + DS_+$	TN	$A' = A - S_\pm$	
/F	$A' = A + DS_\pm$	TF	$A' = -S_\pm$	
/C	$D' = S_+$	TC	$A' = A + S_\pm$	
/K	$D' = S_\pm$	TK	$A' = 2S_\pm$	
/T	B-conditional absolute transfer	TT	$B' = S$	modified
/Z	$S' = H$	TZ	$S' = B$	
/L	Optional stop	TL	$B' = B - S$	
/W	Random number	TW	Ditto	
/H	A-conditional absolute transfer	TH		
/Y	S' = clock	TY		
/P	Unconditional absolute transfer	TP		
/Q	Unconditional relative transfer	TQ		
/O	B-conditional relative transfer	TO	$B' = S$	unmodified
/B		TB	$S' = B$	
/G	Optional stop	TG	$B' = B - S$	
/"		T"	Ditto	
/M	A-conditional absolute transfer	TM		
/X		TX		
/V	Hoot	TV		
/£		T£	No operation	

Instructions are defined by equations in a notation by Turing. The following symbols are used:

A = accumulator
L = lower accumulator
M = upper accumulator
D = multiplicand register
B = B-line
S = store line
H = hand switches
± = suffix meaning signed fraction
+ = suffix meaning positive fraction, no suffix means signed integer
′ = suffix meaning value after operation, unprimed means value before operation

Absolute transfers of control go to line n+1 where n is the "control number" in the line addressed by the instruction. Relative transfers skip n instructions. "A-conditional" means "if the accumulator ⩾ 0" and "B-conditional" means "if the last B-line altered ⩾ 0."

Drum and input/output transfers were specified by a control word (or "magnetic instruction") in the store line addressed by the instruction. The format of this control word has been omitted for brevity.

The usual instructions for setting and performing arithmetic on B-lines were TO, TB, and TG, which were not modified. Occasionally, if modification was required, TT, TZ, and TL were used; these were mainly used for subtle programming tricks.

Peripheral and drum transfers were programmed by a control word known as a "magnetic instruction" or an "auxiliary instruction." It was possible to verify magnetic transfers using special checking instructions provided because of the difficulties experienced on the prototype. Reading, writing, and checking operations took 30, 90, and 45 ms, respectively.

The electronic store was divided into "pages" of 64 lines and each track of the drum held two pages. Transfers between the electronic and magnetic stores always consisted of one or two whole pages. The nature of this "two-level" store affected programming more than any other features of the Mark I.

A 65th line associated with each 64-line page of electronic store contained the number of the track on the drum from which the page had been read. By accessing the 65th line it was possible to associate each page in the electronic store with the track on the drum from which it originated. Kilburn (1979) recalls providing this as a facility that might prove useful to programmers; it was the germ of an idea that later led to the virtual store of the Atlas computer. Sadly, it was not effectively exploited by programmers during the life of the machine, although the engineers found it useful in test programmes for the drum.

Like most early machines, the Mark I was not particularly reliable. The worst problem initially was the magnetic drum, which turned out to have too high a packing density so that some 40% of tracks were inoperative at some point in their life. Much of a programmer's time was spent arranging programmes onto those tracks performing satisfactorily in a particular session. This problem was much alleviated in 1953 when the original 6-inch drum was replaced by one 10 inches in diameter. Similarly, the electronic store was prone to "clodding" by which it picked up spurious digits. It was recommended that the store should not be relied on for more than about 2 minutes of error-free operation. Thus, quite a lot of effort went into making programmes that could be restarted and in which checks were made on the validity of numerical results.

2.2 Programme Organization

The two-level store of the Mark I gave rise to a method of programme organization and use of storage very characteristic of the Manchester school. Vestiges of this can be seen in subsequent Manchester computers, especially in the paged store of the Atlas.

The programming scheme for the prototype Mark I was devised by Turing and he subsequently adapted it to the production model. At the outset a number of arrangements must have seemed possible, but the scheme wherein a programme was broken into pages was a most effective way of using the machine. Certainly, it was much neater than the overlaying techniques generally used when disk stores became available in the early 1960s.

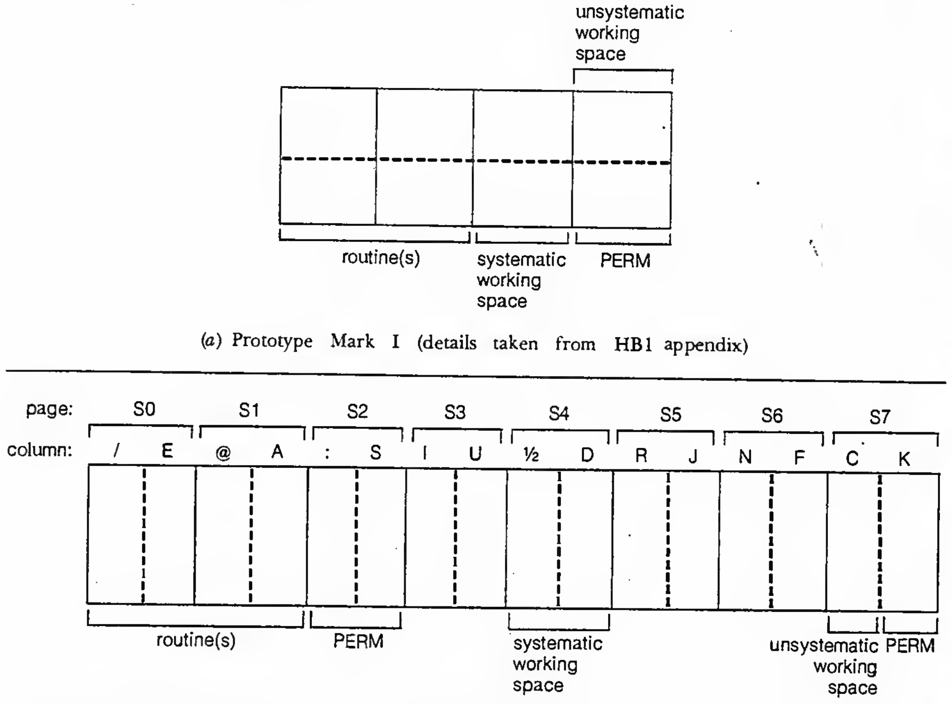
The two levels of storage on the Mark I—magnetic and electronic—constituted a substantial direct-access store and a much smaller immediate-access store, respectively. Turing had a neat analogy for this (HB1, p. 4):

> It is as if information in the magnetic store were written in a book. In order to find any required piece of information it is necessary to open the book at the required page.... The electronic store ... is to be compared rather to a number of sheets of paper exposed to the light on a table, so that any particular word or symbol becomes visible as soon as the eye focusses on it.

The analogy with books and pages is carried to great lengths and the terms *chapter, page, column, paragraph,* and *line* were all used.

Figure 2*a* shows the storage layout of the prototype Mark 1. The electronic store of 128 forty-bit lines was held on four Williams tubes. The memory was divided into four pages of 32 lines, one page per Williams tube. The pages were further subdivided into upper and lower paragraphs of 16 long lines or 32 short lines. (The significance of grouping into 32 short lines will become clear in the next section.) A programme was coded as a series of independent pages or chapters (two pages) that were brought down in turn from the drum to the first two pages for execution. The third page was used for "systematic working space" (i.e., vectors) and the upper paragraph of the fourth page was used for "unsystematic working space" (i.e., variables). Finally, the lower paragraph of the fourth page contained a permanently resident block of code. PERM contained useful constants and a sequence of instructions for bringing down a routine from drum; PERM was the heart of the programming system and had much in common with the resident monitors of later machines.

The physical separation of a programme's instructions and data was done for quite different reasons than in today's reentrant code. It was done on the Mark 1 because a programme was coded as a series of pages that constantly overlaid one another; hence, working storage had to be kept separately.



(*a*) Prototype Mark I (details taken from HB1 appendix)

(*b*) Mark I

Figure 2. Storage organization.

The storage organization on the production Mark I was essentially similar to the prototype. At first 16 pages of electronic store on 16 Williams tubes was planned, but in the end only eight tubes were provided, as shown in Figure 2*b*. The eight pages were designated S0, S1, ..., S7. Each page was divided into two columns and the columns were "numbered" /, E, @, ..., K (for reasons that will become clear shortly). The division of a page into two columns, rather than two paragraphs as on the prototype, reflected the fact that lines were stored column by column instead of row by row. Because of the uncertainty of the eventual size of the electronic store, Turing organized storage assuming five pages, which he regarded as the minimum workable store. This is shown in Figure 2*b*. The remaining pages, S3, 5, and 6, were allocated for additional working space at the discretion of the user. Of course, the designation of storage in this way was purely a convention, and these conventions "should not be regarded as pure tyranny, but to know that they have been obeyed in the programmes one is using is a great comfort" (HB1, p. 48).

2.3 The Written Form of Programmes

In October 1948, at Tootill's instigation, the practice was adopted of using five-bit teleprinter codes to represent binary numbers when reading aloud from the monitor tube so that a second person could write them down (Tootill 1978). Ta-

Table 2. Teleprinter codes

Decimal	Symbol	Binary
0		00000
1	E	10000
2	@	01000
3	A	11000
4	:	00100
5	S	10100
6	I	01100
7	U	11100
8	½	00010
9	D	10010
10	R	01010
11	J	11010
12	N	00110
13	F	10110
14	C	01110
15	K	11110
16	T	00001
17	Z	10001
18	L	01001
19	W	11001
20	H	00101
21	Y	10101
22	P	01101
23	Q	11101
24	O	00011
25	B	10011
26	G	01011
27	"	11011
28	M	00111
29	X	10111
30	V	01111
31	£	11111

ble 2 shows the 32 characters in the teleprinter sequence. A binary number such as

00000 01010 11100 10111

would have been read as /RUX (/, the symbol for zero, was called "stroke"). Teleprinter codes were thereafter used to represent instructions as well. The use of teleprinter codes was established long before the installation of teleprinter equipment in the summer of 1949 and no doubt was a major factor in selecting this type of equipment.

It was perhaps natural (though certainly unfortunate) that Turing should continue to use teleprinter notation in his programming systems for the Mark I, a notation Wilkes (1968) justly described as "bizarre in the extreme."

Because / was used for zero, it was the most commonly occurring symbol in the written form of instructions and gave Manchester programmes a characteristic look (see, for example, Figure 5*d*). At Cambridge, S. Gill remarked (Brooker 1978) that Mark I programmes reflected the Manchester weather, because a programme looked like rain seen through a dirty window pane!

Turing advocated that programmers should begin by learning the teleprinter sequence, which ran

/E@A:SIU½DRJNFCKTZLWHYPQOBG"MXV£

This was easier said than done. Prinz, for one, was driven to devise the following mnemonic (which he remembers to this day):[5]

> stroke Edith at Aberdeen colon She Is Under half
> Dispense Royal Justice Napoleon For Charlie Knows
> Tall Zebras Laugh
> We Hear Young People Question On Best Grounds
> Quotations for a thousand fifteen pounds

The teleprinter sequence thus was indelibly imprinted on the minds of all Mark I users (and those I have spoken to still remember it after a full 20 years). There is a story (Prinz 1978) that Audrey Bates was once asked what letter followed K in the alphabet; she instinctively said T. (Mercifully, the teleprinter sequence was rearranged into alphabetical order for the Mark 1* computers.)

Each 20-bit instruction was written as a sequence of four teleprinter characters. Referring to Figure 1, in the case of the prototype machine, two characters specified the address, one character specified the B-line, and one character gave the operation. There was thus a natural division of the instruction into groups of 5 and 10 bits, each corresponding to one or two teleprinter characters. Unfortunately, in the case of the production Mark I, this natural division no longer obtained, because the operation code had been extended to six bits. Hence, the B-line and the operation code were represented by a pair of teleprinter characters. A particular operation was therefore not represented in the written form of programmes by a unique code; this made programme manuscripts particularly difficult to understand.

The written form of the address part of an instruction looked curious. For example, the address H: would mean line H of column : (a store line was never referred to by its decimal address, line 148 in this case). In fact, because pages and

[5] Tootill's notebook gives a mnemonic that predates Prinz's, but it is not as cute.

columns were so intimately connected with the architecture of the machine, it was rarely necessary to know absolute addresses as such, but instead the line number within a column.

Figure 3 shows a complete instruction for the Mark I. The instruction, written H:YA, stores the accumulator into line H: modified by B-line 5. In a modern assembly language this would probably be written in a form such as

STA 148,5

The written form of Mark I instructions no doubt looks distinctly odd to a modern programmer. But there were some definite reasons for using teleprinter notation, as Turing explained at the Inaugural Conference of July 1951. First, debugging at the console was made considerably easier by having a close correspondence between the written form of an instruction and the binary form in the machine. Second, the programmer needed to be constantly aware of page boundaries, and this would not be nearly so apparent if a decimal address were used. Third, Turing argued, since the names of store lines and function codes were no more than arbitrary labels, it did not matter what they were called.

2.4 The Input Routines

An *input routine* was the term used in Britain for several years for a programme that read in other programmes. Input routines varied in sophistication from the very simplest form of loader at one extreme to systems approaching the complexity of a modern assembler at the other. The input routines for the Mark I were much closer to the former than the latter.

The input routines for the Mark I were permanently stored on the drum on "isolated" tracks. (An isolated track was one to which writing had been disabled so that the track could not be accidentally overwritten.) The input routine was loaded into the electronic store by the following ingenious bootstrapping technique devised by Kilburn and Tootill (Tootill 1978). The electronic store and control registers were cleared and the machine started. It thus obeyed the instruction in location zero, which itself consisted of all zeros (////). The instruction // caused the hand switches to be obeyed as a "magnetic instruction" (see Table 1). Hence, the track designated by the hand switches was brought into the electronic store and control took the next instruction from location I, which now contained the first instruction of the input routine.

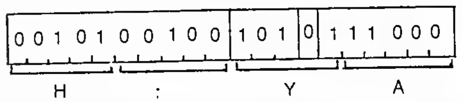


Figure 3. Specimen Mark I instruction.

An input routine was placed on the drum in the first place, or replaced in case of accidental erasure, by a simple bootstrap routine known as "initial input," which copied a binary tape of the input routine onto the drum. The initial input routine was loaded manually by means of the hand switches. (A typical initial input is given in HB3, p. 4-14.)

The input routine for the prototype Mark I was devised by Turing, assisted by D. B. G. Edwards and Tootill, who advised on hardware constraints (Kilburn 1978, Tootill 1978). So far as is known, no copy of the original input routine is extant, but a brief description of it is in the appendix of the first handbook. The input routine was probably first used in October 1949; Popplewell recalls that when she arrived in that month, it was working but only one sub-routine had been written (for reciprocal).

The input routine was simple in the extreme. A user programme was punched onto tape as a series of instruction pairs. Each instruction pair was prefixed by a "warning character" and the address of the long line into which the instruction pair was to be placed. For example, the sequence K/:ABCDEFGH would have put the instructions ABCD and EFGH into the long line/:. The purpose of a warning character (K here) was to distinguish "meaningful sequences" from "intermediate rubbish" such as blank leader tape and erasures. This appears to be the origin of the term *warning character*, which appears frequently in connection with early British software and reappears in Strachey's general-purpose macrogenerator (1965). Once the complete routine had been read into the electronic store, a final meaningful sequence caused the routine to be written on the drum. In this way, each of the routines making up the programme was placed on the drum and finally the master routine was brought from the drum and entered.

It would be difficult to imagine a more primitive input routine than this, in the sense that the conversions from the external symbolic form to the internal binary form were negligible; there was no provision for constants and there were no assembly directives of any substance. Unfortunately, this rudimentary style of input routine was perpetuated on the production Mark I.

There were several input routines for the production Mark I, each having its own PERM and sub-routine library. They are listed in Table 3. The reasons for this evolution can be summarized as follows: Scheme B arose as an improved version of Scheme A; TELEINPUT (or Scheme C) replaced Scheme B when a more reliable magnetic drum was installed; and TELEINPUT/F arose to make better use of the electronic store for certain classes of problem.

The first input routine for the Mark I, Scheme A, was written by Turing and was available about March 1951. A complete listing of the input routine has been located in the Strachey papers, and Section 3 of this paper shows its use for the development of a complete programme.

Scheme A was essentially similar to the prototype input routine, although more flexibility was given by the provision of nine different warning characters instead of the three warning characters in the prototype scheme. Even so, Scheme A provided only the most basic facilities, and programme instructions were still written in teleprinter codes as described in Section 2.2. There was a provision for setting up integer constants, but there is no indication that this was much used: fractional constants had to be converted to base-32 with a Brunsviga calculator. The constant facility was dropped from subsequent input routines.

Scheme A consisted of four pages of code. Because of the generous provision of storage on the Mark I, there was no question of trying to compress the code into the minimum possible space: even so, it is disappointing that such a long routine did so little. For example, at Cambridge a much better input routine had been written in the equivalent of less than a page (Wheeler 1950).

In mid-1951 Glennie arrived from Cambridge, followed in July by Brooker. Very quickly Brooker, in consultation with Glennie, developed a new input routine known as Scheme B. This new routine was working in spring 1952 and ran in parallel with Scheme A. Unfortunately, it proved impractical to make a break with the teleprinter codes, and Scheme B was similar to Scheme A. Users had by now become too entrenched in teleprinter notation to make a change.

Table 3. Mark I input routines

Input routine	Date available	Author
Scheme A	Spring 1951	Turing
Scheme B	Spring 1952	Brooker and Glennie
TELEINPUT (Scheme C)	Early 1953	Brooker
TELEINPUT/F	1954–1955	Gillies

The main advantage over Scheme A was concerned with storing routines on the drum and calling them down to the electronic store for execution. Because the magnetic drum had proved unreliable, the new input routine allowed routines to be numbered serially and allocated actual tracks on the drum by means of a directory. Thus, if a programme could not be loaded because of faulty tracks, a new directory could be quickly prepared and no alteration to the programme proper was necessary. The serial numbering of the routines within a programme shows a strong connection with sub-routine assembly at Cambridge (Wilkes, Wheeler, Gill 1951, pp. 28–32), and the method of calling in a sub-routine, described in Section 2.5, shows similar connections. Incidentally, Scheme A did allow a directory to be used, but the implementation was much less elegant. Scheme B input occupied two pages of code, being more neatly coded than Scheme A.

A new input routine called TELEINPUT, sometimes known as Scheme C, was devised by Brooker probably early in 1953. The main change from Scheme B was that the directory, which had been obligatory in Scheme B, was now made optional. This reflected the improved magnetic store with fewer faulty tracks. It was still recommended, however, that programmes using more than six routines use a directory.

TELEINPUT is the only input routine described in the third handbook, dated August 1953, which seems to indicate that it had replaced Schemes A and B entirely by this date. A comment by Williams (1953, p. 37) seems to indicate that Scheme C and the sub-routine library were completed by about the end of the first quarter of 1953 and that the programming system had at last stabilized.

There was just one further input routine, F/TELEINPUT, by D. C. Gillies. F/TELEINPUT was an extended version of TELEINPUT that allowed routines to be relocated so as to occupy any page of the electronic store. It was mainly used for a limited set of problems where it was desired to keep more than one routine in the electronic store in order to reduce the drum transfer overheads spent in routing changing. F/TELEINPUT was probably written in 1954 or 1955; it was not a replacement for TELEINPUT but was supplementary to it.

2.5 Sub-routines

The concept of the sub-routine was well known in all computing circles (see *Annals*, Vol. 2, No. 1, January 1980, pp. 7–36), but the details of implementation varied considerably, depending largely on the constraints imposed by the computing machine—in particular, the quantity and speed of primary and secondary stores.

The idea of breaking a problem down into a hierarchy of sub-routines also seems to have been commonly understood; in this respect, some of the ideas underlying structured programming are by no means unprecedented. Turing expressed the idea as follows (HB1, p. 60):

> The sub-routines of any routine may themselves have sub-routines. This is like the case of the bigger and lesser fleas. I am not sure of the exact meaning the poet attached to the phrase "and so ad infinitum," but am inclined to think that he meant there was no limit that one could assign to the length of a parasiuc chain of fleas, rather than that he believed in infinitely long chains. This certainly is the case with sub-rouunes. One always eventually comes down to a rouune without sub-routines.

For the Mark I, a programme consisted of a master routine and a group of sub-routines. Normally, each routine was contained in one or two pages and was located on an individual half track or full track of the drum. The programme began execution with the master routine in the first two pages of the electronic store. When a sub-routine was called, it was brought from the drum into the electronic store, overwriting the master routine. The sub-routine eventually terminated by bringing a fresh copy of the master routine and reentering at the instruction immediately following the sub-routine call. Of course, sub-routines could themselves have sub-routines, to any depth.

When a routine was longer than two pages it was divided into pairs of pages, chapters, brought down one at a time and executed in sequence. This was essentially like calling a sub-routine, except that no return link was needed; this was known as calling an "ad-routine."

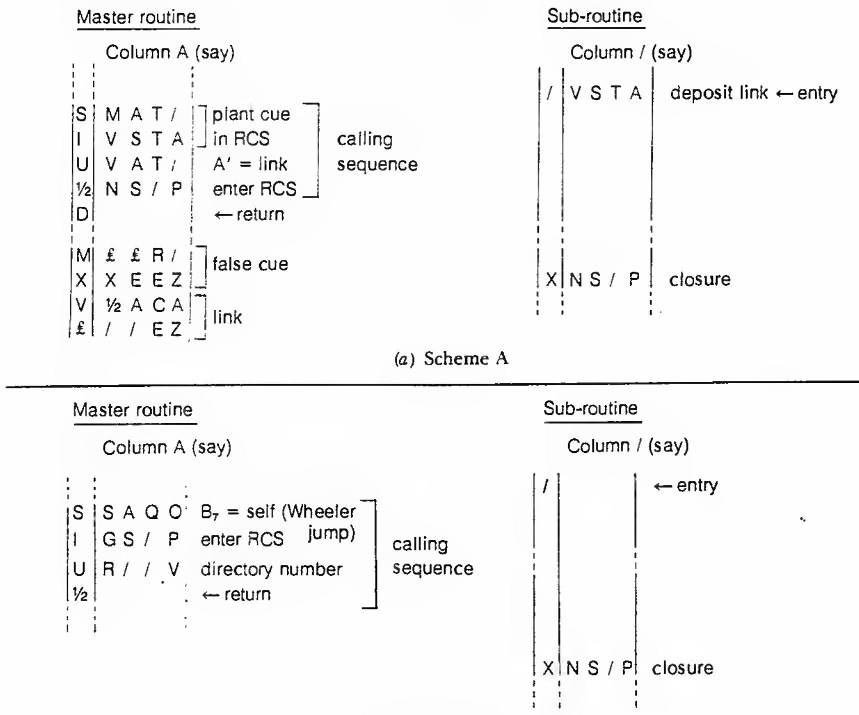
An interesting point of this arrangement was that routines were necessarily confined to one or two pages. Consequently there was very little point, once a routine fitted within a page boundary, in reducing its size further (although considerable effort could be expended on trying to lose the two or three instructions by which the routine exceeded an exact page in size). This was very different in spirit to other installations where users delighted in writing programmes using as few instructions as possible. The drum store of the Mark I was so large that programmers, writing in machine code, were never able to use more than a fraction of it for programme instructions.

Turing's method of routine changing on the prototype Mark I was based on a routine changing sequence (RCS) of five instructions in PERM. Turing devised a similar method for the Scheme A input on the production Mark I, and this is now described.

The basis of routine changing was the "cue." A "true cue" consisted of a long line containing three items: the entry point of the routine, a pair of check digits, and the magnetic instruction to bring down the routine to electronic store. A "false cue" was similar except that a directory was used to hold the true cue and the routine was called down via the directory; this involved an extra drum transfer to access the directory but enabled programmes to be run with an unreliable magnetic store. The directory technique was introduced as a result of difficulties using the prototype Mark I magnetic drum. (The use of the term *cue* appears to be the origin of this term for a library sub-routine call in several early British programming systems.)

Figure 4*a* shows how a sub-routine was called using scheme A. The call is based on a sequence of four calling instructions and two cues. The first cue is used by the RCS to bring down the sub-routine; the second is the link to return control to the calling routine. This routine calling method was unwieldy and suffered from two disadvantages. First, the user had to create a link to return to the sub-routine. Second, the user was responsible for preserving the link in the sub-routine. If the sub-routine called further sub-routines, it was necessary for the user to make private arrangements to stack the links.

Turing's sub-routine linkage was a rather pedestrian piece of work for a person of his talents. This is especially so when we consider that Turing had been aware of this idea of a sub-routine stack as early as 1945 (Carpenter and Doran 1977). Furthermore, Turing would surely have had access to a copy of the "Report on the Preparation of Programmes for the EDSAC and



(*a*) Scheme A

(*b*) Scheme B

Figure 4. Comparison of sub-routine calling methods.

the Use of the Library of Sub-routines" from Cambridge, which was circulated well before the end of 1950 and contained seminal ideas on the use and design of sub-routines.

A much more elegant technique for sub-routine calling was devised by Brooker for use with Scheme B. An example is shown in Figure 4*b*. In this calling sequence, a return link is not explicitly given but is fabricated by the RCS using the technique of the "Wheeler jump," created by D. J. Wheeler at Cambridge (Wilkes, Wheeler, Gill 1951). Similarly, there is no explicit cue for the sub-routine but simply a directory number that gives the location of the true cue in the directory; this had much in common with the technique of sub-routine assembly used at Cambridge. The return link was automatically placed on a stack maintained by the RCS so that sub-routine calls could be nested to any depth and recursion was possible. The sub-routine stack is clearly based on earlier Cambridge work (Brooker & Wheeler 1953).

Parameter passing for sub-routines was restricted somewhat by the fact that a sub-routine overwrote the calling routine; this made it impossible to store parameters following the calling sequence, a technique favored at many installations and pioneered at Cambridge. With Scheme A fractional parameters were placed in a specific store location for the sub-routine to pick up; column C seems to have been the most favored place. For integer and address parameters B-lines were usually used. With Scheme B, the accumulator was free to hold a single fractional parameter, and for sub-routines having a single argument and result this was sufficient; this in fact accounted for most sub-routines.

2.6 *The Sub-routine Library*

When Cicely Popplewell arrived at Manchester in October 1949, her main task was to create a sub-routine library for the prototype Mark I. Unfortunately, little of this early work appears to have survived, although details of the library are given in the appendix to the first handbook. A total of ten sub-routines are named here; half were for input/output and half were mathematical functions. The only remnant of this work apparently surviving is a copy of a reciprocal square root routine attributed to Turing in the private papers of D. G. Prinz, dated November 1949.

Little of this work has survived, probably because the sub-routines were not documented with any great formality; after all, there were only four regular users of the machine. The sub-routine library for the prototype Mark I was never taken to a high degree of completeness because the machine was operational for less than a year, and after a certain point it was futile to develop polished routines for a machine that was soon to be replaced.

The production Mark I was available in February 1951 and the sub-routine library for Scheme A input was quickly started. The first sub-routines to be written were new versions of the sub-routines already prepared for the prototype Mark I. The sub-routines were rigorously documented in what Turing called the "official account," which consisted of a pro forma, known as Programme Sheet I, explaining how to use the sub-routine and Programme Sheet 2 containing a list of the sub-routine. This style of documentation followed the Cambridge pattern used for EDSAC. Each sub-routine was given a name intended to indicate its purpose—for example, DIVISION, LOGFAST, etc. Alternate versions of a routine were suffixed by the letters A, B, C, etc.—thus DIVISION/A, for example.

The Strachey papers contain what is probably an almost complete set of the Scheme A sub-routines. All the routines are dated, but the documentation appears to have been done in bursts—the earliest sub-routine being dated July 1951 and the latest September 1952. There are approximately 50 sub-routines, which might be classified as follows: mathematical 14, input/output 13, and miscellaneous 23. The miscellaneous routines include a number of sub-routines used in connection with the "formal mode" described in Section 2.7; they are clearly the work of Turing.

The authors of sub-routines are not given in the official accounts, but a list of credits for the mathematical routines is given in the second handbook, where sub-routines are attributed to R. A. Brooker, A. E. Glennie, N. E. Hoskin, R. K. Livesley, C. M. Popplewell, and A. M. Turing. Sub-routines were placed in the library when submitted by users and when it was felt they were of general use. Popplewell prepared the documentation and was responsible for testing the routines.

A second sub-routine library was prepared for Scheme B users in 1952. Where possible sub-routines were made compatible with both Scheme A and B, and this was achieved for at least half of them. The Strachey papers contain only a few of the early Scheme B sub-routines, presumably because he had by now left for Canada. Much of the Scheme B library would thus appear to be lost. The second handbook and its supplement indicate that there were about 20–30 sub-routines. In addition to the basic library of sub-routines, there was an excellent interpretive floating-point scheme, FLOATCODE, described in Section 4.1. Much of the sub-routine library was "exported" to the University of Toronto, where it formed the basis of the library for the FERUT, the second Ferranti Mark I (Gotlieb 1954, p. 125).

A third sub-routine library was developed for Scheme C, but there was now a much clearer idea of exactly what sub-routines were needed. Brooker said (1956, p. 152):

> The number of library routines has been kept as small as possible by ruthlessly excluding routines of marginal utility. Thus, no attempt has been made to provide alternative versions in which, for example, emphasis is laid on time or space economy measures. Instead, those that are available are described in detail, so that a user can modify it if he thinks this desirable.

The third edition of the handbook contains what was presumably the complete set of 18 library sub-routines as of September 1953, made up as follows: mathematical functions 6, input/output 4, and floating point 8. A compound library tape contained the entire library and could be input to the drum in about half a minute. This was only possible because there was a fast paper-tape reader, a large backing store, and a very small library. Programmers generally used the compound tape rather than composing their own programme tape from the individual routines; this was a considerable convenience. In addition to the sub-routines, the library also contained a small number of complete programmes of general utility. These included programmes for matrix work, Runge-Kutta, quadrature, etc.

A final picture of the sub-routine library, which grew considerably from the original 18 sub-routines, is given in the supplement to the third handbook (dated January 1956). This contains a large number of library routines, as follows (with their authors): print routines, Bessel functions (B. Richards); double-length routines, quadrature routines (C. M. Popplewell); complex functions (M. P. Hyde); matrix routines (F. H. Sumner); analytical differentiation (D. Morris).

2.7 *Turing and the Formal Mode*

One of the most interesting "finds" in this investigation into programming activity at Manchester was an elaborate protocol for running programmes by Turing, which he called the "formal mode." It would perhaps not be overstating the case to describe this as an operating system in embryo.

The principle of the formal mode was to provide well-defined interfaces between programmer, machine, and operator. The first handbook (HB1, p. 55) noted:

> The advantage of working in the formal mode is that the output recorded by the printer gives a complete description of what was done in any computation. A scrutiny of this record, together with certain other documents[,] should tell one all that one wishes to know. In particular this record shows all the arbitrary choices made by the man in control of the machine, so that there is no question of trying to remember what was done at certain critical points.

This rigid protocol seems strangely at variance with Turing's own methods of working, which tended to be unorthodox and unregimented. It seems likely that Turing may have been influenced by his wartime experience at Bletchley Park. D. Michie (1978) writes that Turing's formal mode is "strongly reminiscent of the kinds of routines and administrative operations we followed in the machine room ... when the use of the COLOSSI was going on a regular shift basis." In fact, Turing probably picked up the operational style for his formal mode in a different group than Michie's, but the regime would have been very similar.

The heart of the formal mode of operation was a sub-routine called ACTION, which was the interface between the programmer and operator at run-time. The rule was that a programme would only issue operator instructions by passing a message through ACTION; likewise, the operator would only respond to a message if it had been produced by ACTION. Such messages took the form

++ACTIONmessage--

After performing the action, usually loading a tape or setting the hand switches, the operator would restart the machine; ACTION would then print out the hand-switch setting, thus producing a record of the machine state at each step of the job. ACTION was a very short and simple routine (less than a page), and its sole function was to provide a formal channel for, and record of, communication between programme and operator. The original ACTION sub-routine has been lost, although there are later versions written in 1951 and 1952 in the Strachey papers.

Associated with the formal mode were a number of what we would now call "system routines." One of these, INTERFERE, enabled the operator to alter single lines in the store at the console, not unlike a present-day on-line debugging routine. Another more interesting routine was used with the "bursts" procedure. Because of the unreliability of the Mark I, it was usual to perform any long computation in several bursts of a few minutes each; as an additional check on reliability, each burst was repeated until it gave the same result twice. The sub-routine BURSTS B formalized this arrangement so that a sub-routine could be performed repeatedly until consistent results were obtained, as determined by printing a check sum of the magnetic store. An associated routine COPY R OUT could be used to produce a reloadable dump of the magnetic store at the end of a burst; this is a very early example of a dump and restart procedure.

Sadly, much of the ingenuity of the formal mode was wasted, because Turing was probably the only person who used it, and it was largely ignored by the other users. Two reasons can be identified for this. First, there was no operator for the Mark I at this time (even for Turing), so the formal mode was really something of an academic exercise. Second, programmers tended to debug and run programmes in a totally ad hoc manner, and the formal mode would have been a fussy and unnecessary barrier. Nevertheless, Turing made extensive use of the formal mode for his own programmes, and some examples of his programme output are in his papers at King's College, Cambridge. Other aspects of the formal mode, such as conventions for preparing programme tapes, were bound up with Scheme A input and were used by all programmers until Scheme B arrived. The following section illustrates some of these aspects.

3 A Tutorial on Programming the Mark I

In order to illustrate the material in Section 2, this section develops a complete programme for the Mark I, as might have been done early in 1951 when the production machine had just come into use. The programme is developed using Turing's Scheme A and its associated sub-routine library.

To provide a concrete example by way of illustration, a programme known as TPK has been prepared for this section, detailed in Figure 5. (An explanation of the TPK programme is given in the Appendix.) Figure 5 is primarily intended to convey the flavor of programming for the Mark I—for example, what the coding looked like and how the results would have appeared. Many readers, especially at a first reading, will wish to treat Figure 5 in this spirit. A deeper study of Figure 5 will reward the interested reader with a number of small but intriguing programming details. The TPK programme has been tested on a simulator for the Mark I, so the coding and the results may be regarded as reliable.

3.1 *A Programme for the Mark I*

In the first handbook (HB1, p. 51) Turing defined programming as "an activity by which a digital computer is made to do a man's will, by expressing this will suitably on punched tapes." He itemized four stages of programming, used in the text below:

1. Make a plan.
2. Break the problem down.
3. Do the programming of new sub-routines.
4. Programme the main routine.

Make a plan.

> This rather baffling piece of advice is often offered in identical words to the beginner in chess. Likewise the writer of a short story is advised to "think of a plot" or an inventor "to have an idea." These things are not the kind that we try to make rules about (HB1, p. 61).

For the most part planning the problem involved doing the numerical analysis and scaling. The Mark I held fixed-point signed fractions in the range $-\frac{1}{2} \leq x < \frac{1}{2}$. Figure 5*a* shows the scaling calculation for the TPK programme that ensures that all numbers stay within range. Scaling was usually the most difficult part of planning a programme. For those who were less gifted numerically, and for small or one-off problems, floating-point routines were available to obviate the scaling problem at a cost of increased execution time. Machines such as the Mercury with hardware floating point were eventually developed.

The next task is to plan the use of the store. There is little choice here but to follow the normal pattern for Scheme A: see Figure 5*b*.

Break the problem down.

> This in effect means to decide which parts of the problem should be made into definite sub-routines. The purpose of this is partly to make the problem easier to think about; the remaining work consisting of a number of "limited objective" problems (HB1, p. 63).

Breaking the problem down thus consisted of dividing the programme into a master routine and sub-routines. So far as possible sub-routines would be taken from the library. In addition, any "auxiliary" sub-routines would be coded especially for the programme. Figure 5*c* lists the routines for TPK.

Do the programming of the new routines. Turing advocated programming the sub-routines before the main programme—that is, bottom up instead of top down. This was mainly because it was generally necessary to fix the details and scale factors of a sub-routine before a higher-level routine could be coded.

Figure 5*d* shows the coding for TPK. It is probably just about feasible to follow the detailed coding from the information in this paper, but not very profitable. It should be possible, however, to follow the general flow from the accompanying comments, which are intended to be in a format similar to that used on actual programmes. Notes have been added to explain some of the finer points of programming for the Mark I.

Programmes were normally coded directly onto coding sheets. The coding sheets consisted of quarter-inch-squared paper overprinted with the teleprinter sequence arranged in the two columns making up a single page routine. Some of the details of annotation (for example, underlining unconditional transfer-of-control instructions) and the style of comments followed the pattern used on the EDSAC at Cambridge. When the programme was punched, however, only the instructions themselves were transcribed.

The text continues on page 152, after completion of Figure 5.

In the TPK algorithm for each element t in the vector we have to calculate

$$y = \sqrt{|t|} + 5t^3 \qquad (1)$$

if we assume t is less than about 10 in magnitude then we can rewrite equation 1 as

$$y' = 2^{-13}\sqrt{2|t'|} + 5t'^3 \qquad (2)$$

where $y' = 2^{-15}y$ and $t' = 2^{-5}t$. Now all numbers are less than one half in magnitude.

(*a*) Scaling calculation

Pages	Contents
S0. S1	ROUTINES
S2. S7 col K	PERM
S4 col ½	VECTCR (systematic working space)
S7 col C	VARIABLES and SUB-ROUTINE PARAMETERS (unsystematic working space)

(*b*) Storage organization

Routine	Description
MASTER	Corresponds to the main body of the TPK algorithm
AUXILIARY	Corresponds to the procedure f in TPK
INPUT/A	Library routine, reads a vector
DEC-OUTPUT	Library routine, prints a fraction
RECIPROOT	Library routine, reciprocal square root

(*c*) Table of routines

Notes to Figure 5d, master routine

The master routine occupies 2 pages, of which approximately 1½ pages are used. No attempt has been made to economize on the space taken.

[1] The original TPK algorithm uses a for-loop to read the vector. There was a library routine, INPUT/A, to read a vector, however, so it is used here.

[2] Cycles around the main loop by setting B_4 to 20 and reducing by two each time around the loop until it gets to zero. The B-line will also be used to access the array elements stored in long lines—hence the decrement of two.

[3] There was no sub-routine to print an integer, so some awkward scaling operations are used here to adjust the number for output as a fraction.

Figure 5. TPK for the Mark I.

Column /

```
/ | " E P O | B_6 = H½·pointer
E | T @ T / |                      read array
@ | V S T A | plant cue            values using
A | L @ T / | A' = link            INPUT/A[1]
: | N S / P | enter RCS
S | E @ H O | B_4 = 20[2]
I | X E / : | new line
U | V E / : |
½ | @ @ H B |
D | I @ T / | scale B_4
R | I @ / C |   by
J | @ @ / N | 2^39·10^-2
N | T C T A |                      print ½ · B_4
F | M E Y O | set parms            using
C | A : P O | for                  DEC-OUTPUT[3]
K | B K T B | DEC-OUTPUT
T | H @ T / |
Z | V S T A | call
L | P @ T / | DEC-OUTPUT
W | N S / P |
H | £ E / : | two spaces
Y | £ E / : |
P | / / T : |
O | ½ @ / K |
O | / ½ : F | [5] scale a_i
B | T C / E | by 100·2^-5
G | / / T : | and save
" | U : / K | in TC[4]
M | T C / F |
X | T C T A |
V | M @ T / | call AUXILIARY
£ | V S T A | using TC . . .
```

[4] The routine INPUT/A only accepts values in the range $-\frac{1}{2} \leq x < \frac{1}{2}$ so the data had to be scaled by 10^{-2} (for example, 0.099 was used for 9.9). Hence the scale factor of $100 \cdot 2^{-5}$ rather than 2^{-5}.

[5] This is the only instruction in the master routine that uses modification. It picks up the next element of the vector.

Column E

```
/ | V @ T / | . . . for argument t'
E | N S / P | and result y'
@ | R @ T / |
A | T C T N | [TC] becomes y'
: | A : / M | if y' < 400·2^-15
S | N @ T / | otherwise 999·2^-15
I | T C T A |
U | / / T : |
½ | : @ / J | scale [TC]
D | C @ / K | by
R | T C / F | 10^-5·2^15
J | T C / E |                  print
N | M E Y O | set parms        [TC] · 10^-5 · 2^15
F | / @ P O | for              using
C | B K T B | DEC-OUTPUT       DEC-OUTPUT
K | O @ T / |
T | V S T A | call
Z | G @ T / | DEC-OUTPUT
L | N S / P |
W | A : H G | B_4 = B_4−2
H | G E / T | branch to I/ if B_4 ≥ 0[2]
Y | H E / V | hoot stop[6]
P | Y E / P |
O | / / / / |
O | / / / / | spare
B | / / / / |
G | S / / / | control number
" | H ½ / / | pointer to line H½
M | T C / / | pointer to line TC
X | / / L / | carriage return
V | / / W / | line feed
£ | / / Z / | space
```

Column @

```
/ | R / / / | 10
E | H / / / | 20
@ | / / / / | copy of B_4
A | / / / / |
: | O E / / | ½·10^-10 (rounding constant)[7]
S | / / / / |
I | G A S M | 2^39·10^-2
U | G A S / |
½ | / / / / | 100·2^-8
D | / / T N |
R | / / / / | 400·2^-15
J | / T N / |
N | / / / / | 999·2^-15
F | / U £ / |
C | C M ½ N | 10^-5·2^15
K | F Z K R |
T | / / R / | cue
Z | X E E Z |        INPUT/A
L | : / C A | link
W | / / E Z |
H | L E " : | cue
Y | A E E Z |        DEC-OUTPUT[8]
P | W / C A | link   entry 2
O | / / E Z |
O | / / " : | cue
B | A E E Z |        DEC-OUTPUT
G | L E C A | link   entry 1
" | / / E Z |
M | £ £ ½ C | cue
X | E / E Z |        AUXILIARY
V | E E C A | link
£ | / / E Z |
```

(*d*) Programme—master routine

[6] The hoot stop causes the hooter to be pulsed each time around the two-instruction cycle. This gave a continuous note until the machine was stopped.

[7] Most of the constants were obtained from an appendix in the second handbook, attributed to the FERUT group. Powers of. two, used at several points in the programme, are obtained from PERM.

[8] DEC-OUTPUT has two entries. Entry number two is used for positive fractions and suppresses the sign.

Figure 5 continues overleaf.

```
Column /

/   V C T A     save link in VC[1]
E   T C T K  ]
@   E : / M  ]  [/C]' = |2t'|
A   D S T J  ]
:   / C T A  ]
S   : E T /  ]                      ]
I   V S T A  ]  call                ]
U   I E T /  ]  RECIPROOT           ]  form
½   N S / P  ]                      ]  2^-13 √[/C]
D   / / T :  ]                      ]  in @E
R   I C / C  ]  destandardize       ]  using
J   I A S N  ]  and form root       ]  RECIPRCOT[2]
N   @E / A   ]                      ]
F   / / T :  ]
C   T C / K  ]
K   T C / F  ]
T   / E / E  ]
Z   / / T :  ]  A' = 5t'^3
L   / E / F  ]
W   / E / U  ]
H   / E / S  ]
Y   S : / K  ]
P   / E / F  ]
Q   @E T C   ]  [TC] = 2^-13 √|2t'| + 5t'^3
O   T C T A  ]
B   V C T /  ]  plant link          ]
G   V S T A  ]                      ]  closure[3]
"   N S / P     enter RCS           ]
M   / / / /  ]
X   / / / /  ]  spare
V   / / / /  ]
£   / / / /  ]
```

```
Column E

/   U S E D  ]  working storage[4]
E   U S E D  ]
@   U S E D  ]  working storage
A   U S E D  ]
:   £ £ G A  ]  cue     ]
S   ½   E Z  ]          ]  RECIPROOT
I   ½ : ½ C  ]  link    ]
U   E .  E Z ]          ]
```

Notes to Figure 5d, auxiliary sub-routine

The auxiliary sub-routine occupies one page. It evaluates (2) using location TC for the argument t' and the result y'.

[1] Because AUXILIARY uses a sub-routine we have to preserve the retum link somewhere safe.

[2] RECIPROOT is Turing's combined reciprocal and square root sub-routine. By calculating the reciprocal of the square root, both reciprocal and square root could be formed by the relations

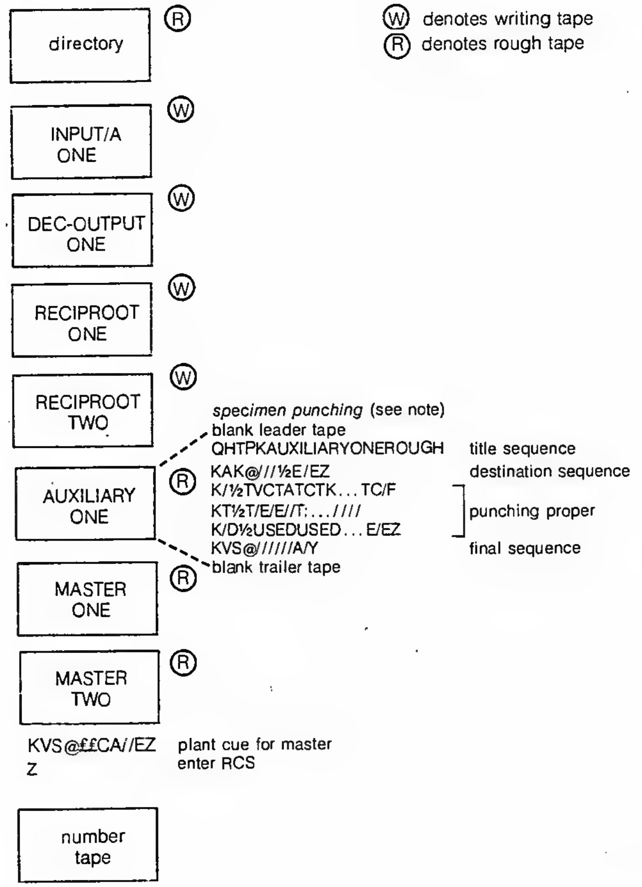
$$(1/\sqrt{N})\cdot(1/\sqrt{N}) = 1/N \text{ and } N\cdot(1/\sqrt{N}) = \sqrt{N}$$

To give maximum accuracy the result is "standardized" (i.e., the most significant digit is a one). The destandardization, and formation of reciprocal or square root, is achieved by a special piece of code given in the official account.

[3] To retum to the master, we have to restore and plant the link, and then enter the RCS.

[4] There was a convention that local working storage in a routine was filled with the word USED, as a kind of comment.

(*d*) Programme—auxiliary sub-routine



```
specimen punching (see note)
blank leader tape
QHTPKAUXILIARYONEROUGH
KAK@///½E/EZ
K/½TVCTATCTK...TC/F
KT½T/E/E//T:..../////
K/D½USEDUSED...E/EZ
KVS@//////A/Y
blank trailer tape
```

```
KVS@££CA//EZ
Z
```

(*e*) Makeup of programme tape

```
++INITIAL--

ODDIRECTORY
ORINPUT/AONE
QNDECOUTPUTONE
ONRECIPROOTONE
ONRECIPROOTTWO
OHTPKAUXILIARYONEROUGH
QZTPKMASTERONEROUGH
QZTPKMASTERTWOROUGH
10  +0000004472
09  +0000003162
08  +0099900000
07  -0004485585
06  +0004775414
05  +0099900000
04  +0006235157
03  +0099900000
02  -0107755050
01  +0099900000
00  +0001809974
        ↑
        └decimal point
```

(*f*) Printed output

Note to Figure 5e

In addition to the punching, there were various waming characters and special sequences: the title sequence labels the tape "TPKAUXILIARYONEROUGH," the destination sequence associates the routine with an entry in the directory, the punching proper causes the routine to be assembled in the electronic store, and the final sequence transfers the routine to a half-track of the magnetic store. The directory and master routine would be punched similarly.

Figure 5 continued.

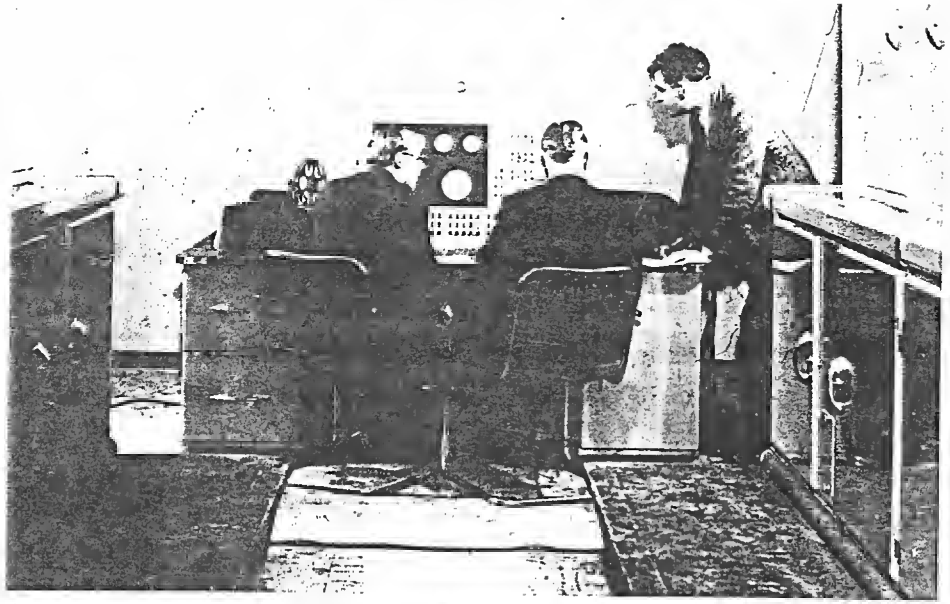


Figure 6. A general view of the Ferranti Mark I at Manchester University. Logic cabinets are to the left and right of the picture, and the control desk is in the center. The figure standing is A. M. Turing.

3.2 *Punching and Running*

The next job would be to punch the master and auxiliary routines and directory. For Scheme A, there were two types of routine tapes—rough tapes and writing tapes. A rough tape was the first form in which a routine was punched prior to testing. Once a routine had been checked, a utility could be used to punch a writing tape. The writing tape was a cleaned-up version of the rough tape and included a check sum. Library routines and production programmes were kept in the form of writing tapes.

Rough and writing tapes were key features of the formal mode of use. Rough tapes for routines were punched in a rigid format, and Figure 5*e* includes the punching of one routine as an example. Tapes were blind punched on one of two keyboard perforators, programmers usually punching their own tapes.

Sub-routine tapes were kept in small boxes in a set of storage drawers. The lowest drawer contained the master copies, punched on blue tape. There was one routine tape for each page of the routine. Using the reperforator, the various routines were usually duplicated to produce a single continuous programme tape. Library routines were then rewound and returned. In fact this was not essential, and an alternative was to input the individual tapes one at a time. Handling so many individual tapes wasted time and could be a source of errors, however, so composite tapes were usually used. Figure 5*e* shows the makeup of a complete programme tape for TPK.

Figure 6 shows a general view of the Mark I machine room. The machine was normally booked for at least an hour. To run the programme, the programme tape was placed in the reader. For Scheme A the hand switches were reset, store cleared, and the machine started; this brought down a starter routine that heralded its arrival by printing ++INITIAL−− (see Figure 5*f*). As each routine was input, the titling sequence was copied onto the teleprinter. thus producing a printed record of the input process. Input for TPK would have taken about one-half minute.

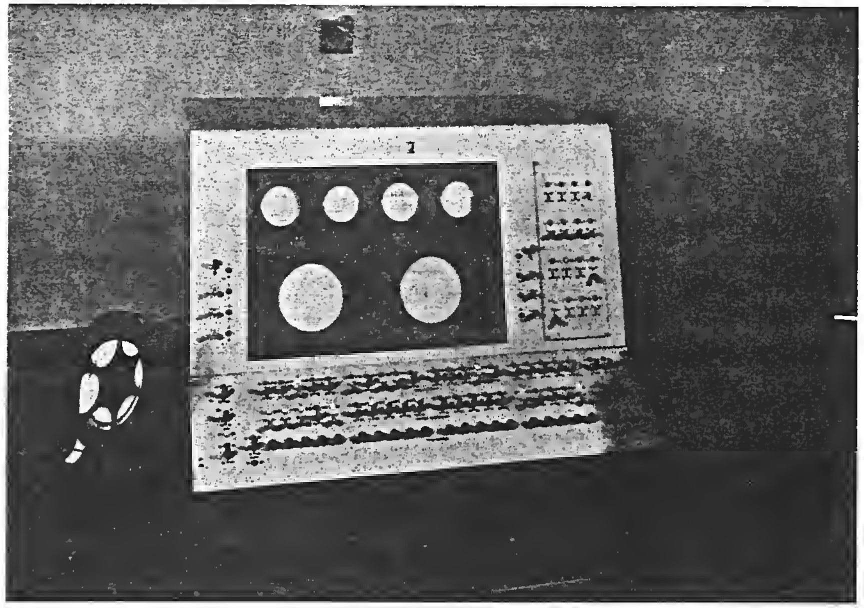


Figure 7. The two large (6-inch) monitor tubes displayed any two tubes in the electronic store on the Mark I console. The four 3-inch tubes above monitored the accumulator, B-lines, and control registers. The hand switches could be used to patch programmes during debugging. The paper-tape punch and reader are to the left of the console, and the teleprinter is on the right.

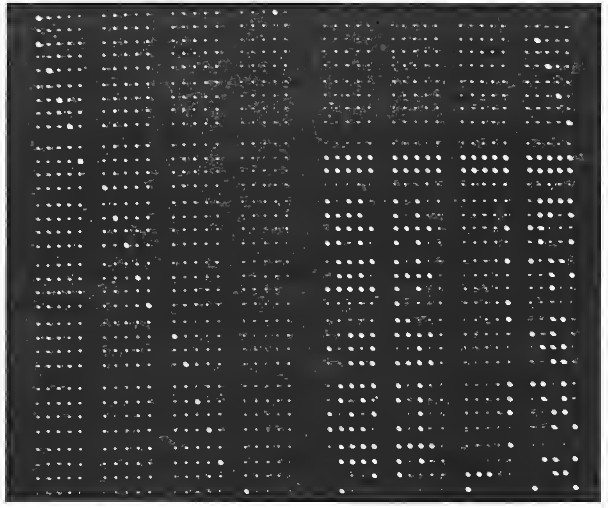


Figure 8. The Mark I monitor tube displayed the two pages held in a Williams tube. The store lines are divided into five-bit groups corresponding to the teleprinter characters in the written form of the programme. The code displayed in this instance is the Scheme A PERM resident in page 4 of the electronic store. The 65th line, top left, indicates the drum track from which the pages were obtained.

The programme would then be automatically entered, Figure 5*f* shows the printed results. Programme execution would have taken about a further half minute, the whole job taking a little over one minute in all. Finally, the programme would terminate on a "hoot" stop (a programme loop containing a hoot instruction; this produced a continuous note "about middle C" signaling the end of the programme). Incidentally, the loudness of hooting could be set by a volume control. Popplewell recalls that hoot stops went out of fashion when a new user ran a programme that terminated on a hoot stop, with the volume full on. He looked for a switch to turn off the hooter and the only one that seemed possible was labeled HT. Unfortunately it was the high-tension power supply; the machine took several days to repair!

3.3 Programme Checking

Programmes rarely worked the first time they were tried—even by the best programmers. Strachey (1966) expressed this rather well:

> Although we are happy to pay lip service to the adage that to err is human, most of us like to make a small private reservation about our own performance on special occasions when we really try. It is somewhat deflating to be shown publicly and incontrovertibly by a machine that even when we do try, we in fact make just as many mistakes as other people. If your pride cannot recover from this blow, you will never make a programmer.

Debugging was not a term in use in the early 1950s in Britain, where the practice was known as *programme checking*. Figure 7 shows the console of the Mark I, which had exceptionally good monitoring facilities and aids for programme checking. These included a pair of 6-inch monitor tubes that enabled any two tubes in the electronic store to be displayed. Figure 8 shows a close-up of one of the monitor tubes. The ease with which monitor tubes could be used with CRT storage was a very attractive feature of this technology; there is no equivalent for today's memories.

By sitting at the console, the programmer could observe the progress of the programme on the monitor tubes in a process known as "peeping." Peeping was very much the modus operandi at Manchester; this attitude was very different from that at Cambridge where the practice was strongly discouraged as being wasteful of machine time. This was one of the most significant differences between computing at Manchester and Cambridge: at Manchester the machine and its operations were very much geared to on-line programme checking, whereas at Cambridge the operating regime encouraged correcting "blunders" well away from the machine. It was some years before Manchester had a machine operator.

Turing advocated the use of "checksheets" in association with peeping. The principle of checksheets was that before running a programme, the author would dry-run the programme, entering on a checksheet the anticipated contents of the accumulator and relevant store lines as each instruction was obeyed. Programme testing then simply consisted of "single prepulsing" through the programme and comparing the actual with the predicted behavior. Once a mistake was found, a correction could be inserted there and then, or the programmer could finish and ponder the error at leisure. Checksheets were a laborious way of testing a programme, and Turing was the main exponent of this particular technique, although other programmers used them on occasion.

Besides single prepulsing, a programme could be monitored at full speed, although the execution rate of about 850 instructions per second was too fast to follow the programme in detail. An interesting innovation was a third alternative by which instructions were obeyed at the rate of 50 per second. A characteristic of the CRT store was that as an instruction or operand was fetched the corresponding line on the monitor momentarily brightened; thus, working in "slow-motion" mode, a programme loop, for example, could be observed as a bright band repeatedly moving down the page.

The Mark 1 had two "optional stop" instructions that could be made operative by setting either or both of a pair of console switches. When a programme had been fully developed the stops were disabled and the programme would run at full speed. The use of stops enabled checked-out parts of a program to be run at full speed; a stop placed at the beginning of a critical section then enabled it to be checked instruction by instruction. The routine changing sequence contained a stop instruction so that the machine could be made to pause after each routine change, allowing tested routines to be run at full speed and new routines monitored.

A device occasionally used was to place hoot instructions at strategic points in the programme. This produced a musical sound indicating when one part of the programme had finished and another part started—and also when something went wrong. The sound of a programme could be

quite pleasant and Prinz (1978), for example, recalls once being congratulated on the musicality of one of his programmes.

Checking routines of the type pioneered on the EDSAC at Cambridge (Gill 1951) were little used at Manchester for two reasons. First, the powerful monitoring facilities made peeping a more attractive alternative. Second, the use of checking routines interfered with the conventions concerning the use of the electronic store. An interpretive checking routine (trace routine) by Glennie printed out the function code letter of each instruction as it was obeyed, but this routine (developed in September 1951) was shortly made obsolete by the "development switch" added in 1952. The development switch produced an automatic listing of the function letter of each order as it was obeyed; this could be switched on or off at the discretion of the operator. The D-switch was not often used, in fact, because the information it yielded was not particularly useful; the really difficult errors were those concerned with numerically incorrect answers (Brooker 1956, p. 154). The second handbook contains an interpretive checking routine that provided numerical trace information, but again it was probably little used.

Postmortem routines were also little used at Manchester. The reason for this is that there was no redundancy in the order code. Consequently, if a programme went wrong and control was transferred to lines containing data, the machine would grind on remorselessly obeying nonsense instructions. An even worse feature was that empty words caused the hand switches to be obeyed as a magnetic instruction; this could result in the input routine being brought down to the electronic store, overwriting the original routine in the process, unless the hand switches had been carefully reset before entering the programme.

It is interesting that once the practice of checking by peeping had become established—and the console and programming system had been designed with this in mind, of course—there was little chance that alternative methods could compete.

4 Simplifying Programming

Programming the Mark 1 was no simple matter, and a good deal of effort went into trying to make things easier for the programmer. Three major difficulties were associated with programming the Mark I:

1. *Scaling.* It is difficult today to appreciate that probably the biggest problem facing programmers in the early 1950s was scaling numbers so as to achieve acceptable precision from a fixed-point machine. In big programmes there could be several different scale factors in order to maintain precision at different stages of the computation. Brooker recalls that whether machines should have floating-point hardware or not was probably *the* issue in computing circles in the early 1950s. But until suitable hardware came along, floating point had to be programmed.

2. *The two-level store.* A real difficulty with the Mark I was organizing the flow of information between the magnetic and electronic stores. This was not too difficult for programmes, because they naturally tended to break into a hierarchy of routines. But for data the problem could be quite severe; for example, matrices had to be partitioned into page-size chunks and brought down one piece at a time.

3. *The language difficulty.* Programmes written in the teleprinter code were difficult to write and even worse to understand. Because there was no relocation or symbolic addressing, programme corrections tended to be done by out-of-line coding, which was always a potential source of trouble. As Glennie put it (1952*b*, p. 2): "It is quite difficult to decipher coded programmes even with notes and even if you yourself made the programme several months ago."

In this section we examine the three most significant attempts to overcome these problems: FLOATCODE and the autocodes of Glennie and Brooker. In each case TPK will be coded for the system. Original listings of FLOATCODE and Brooker's autocode have been located and used on the Mark 1 simulator, so the versions of TPK given are working programmes. Unfortunately, this has not been possible for Glennie's autocode.

4.1 An Interpretive Floating-Point Scheme

Probably the earliest interpretive floating-point scheme for the Mark I was by D. G. Prinz of Ferranti, an external user. The scheme was developed during 1951 and was described at the 1952 National Computer Conference, Toronto (Bennett et al. 1952). This scheme was apparently used only by the Ferranti group.

The most useful, and used, interpretive floating-point scheme was FLOATCODE by Brooker, which ran under Scheme B. The official accounts of FLOATCODE are dated July 1952, and a full description is given in the second handbook. The basis of FLOATCODE was a generalized interpretive routine called INTERCODE, which could be used as the nucleus of any interpretive scheme, although in fact it does not appear to have ever been used for any other scheme than FLOATCODE. INTERCODE was essentially a switchbox that decoded interpretive instructions and transferred control to various entry points in the main interpretive routine.

Figure 9*a* shows the storage layout for FLOATCODE. The interpretive scheme was permanently resident in four pages. Two pages were given over to the user routine and two pages to working space. Figure 9*b* shows the coding for the TPK programme. Interpretive instructions were written in the same format as normal instructions and were input by the normal input routine, the only difference being that the interpretive instructions were obeyed by FLOATCODE rather than by the control circuits of the computer. It was possible, and necessary, to mix normal and interpretive instructions. The printed output from the programme is shown in Figure 9*c*.

Floating-point arithmetic was slow compared to normal fixed-point operations; for example, addition took 90 ms compared to 1.3 ms. On the other hand, a large part of any programme was concerned with organizational instructions, drum transfers, and input/output, which were unaffected by interpretation. Further, most of the floating-point library routines, such as square root, were written in normal instructions and were just as fast as the fixed-point equivalents. For most problems the speed penalty would have been no more than a factor of ten, and often less. The Mark I simulator indicated a factor of about two for the TPK programme.

FLOATCODE shows very clearly its origins in the floating-decimal scheme Brooker developed with Wheeler (1953) at Cambridge shortly before arriving at Manchester. Much of the terminology and organizational details are the same, although the implementation is much more polished in the Manchester scheme.

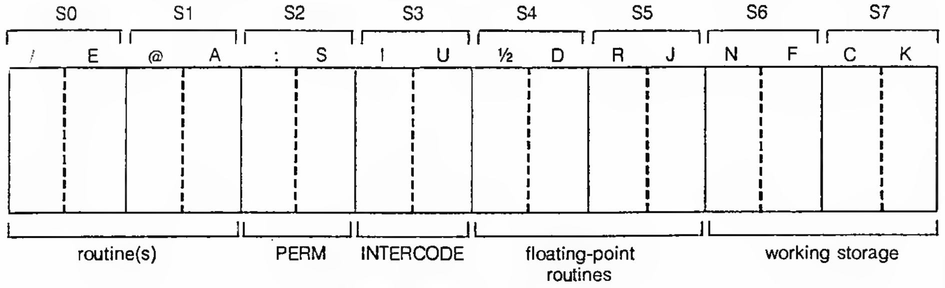
FLOATCODE was quite heavily used for jobs where speed of execution was of secondary importance, and Popplewell recalls that it was one of the really useful routines. FLOATCODE seems to have been eclipsed in 1953 when Scheme B was replaced by Scheme C. For Scheme C a new set of floating-point routines was provided. Although the new scheme was sometimes described as interpretive, this was not in fact the case. Operations were effected by a series of calls on a set of closed routines and there was no interpretive code as such. Although this was much less elegant than FLOATCODE, this was of little importance; with the arrival of the Mark I Autocode in 1954 floating point was generally done via the autocode rather than directly.

4.2 *An Experiment in Automatic Coding*

In 1952 Glennie devised an automatic coding system to try to alleviate the language difficulty of programming the Mark I. Glennie called the system AUTOCODE;[6] this appears to be the origin of the term, which was widely used in Britain in

[6] In general, the spelling *AUTOCODE*, used by Glennie, and *Autocode*, used by Brooker, will be adopted when referring to their respective systems. The generic term for all such systems will be *autocode*, which follows the normal usage in the early literature.

The text continues on page 159, after completion of Figure 9.



(*a*) Storage organization—floatcode

Figure 9. TPK in FLOATCODE.

```
     Column /

/ | / / Q O |  enter FLOATCODE¹
E | E I / P |
@ | / / / V |  read constants²  [ 999 in /F
                                   400 in @F
                                     5 in : F
A | / / / V |  read array*
: | : / / £ |  leave FLOATCODE³
S | U E H O |  B₄ = 20
I | S E / : |  new line
U | I E / : |
½ | / / T : |
D | ½ E H B |
R | I S / C |
J | ½ E / N |  print ½ · B₄
N | @ C / E |
F | F / £ O |  using DECOUTPUT/A
C | E I / P |
K | I / / V |
T | T / Q O |  reenter FLOATCODE
Z | E I / P |
L | / C / Y |  F′(A) = aᵢ⁴
W | / N : W |
H | ½ / / V |  call AUXILIARY⁵
Y | @ C / Y |  copy y into @C
P | @ C / W |
Q | @ F / H |  F′(A) = F(A) − 400
O | " / / M |  branch if F(A) positive
B | / C / Y |  F′(A) = y
G | @ C / W |
" | X / / " |  branch
M | / C / Y |  F′(A) = 999
X | / F / W |
V | @ / / V |  print F(A) using FC:DECPRINT
£ | £ / / £ |  leave FLOATCODE

* into column N, lines /N . . . HN
```

```
     Column E

/ | A : H G |  B₄ = B₄ − 2
E | : E / T |  branch to I/ if B₄ ≥ 0
@ | E E / V |  hoot stop
A | @ E / P |
: | S / / / |  control number
S | / / L / |  carriage return
I | / / W / |  line feed
U | H / / / |  20
½ | / / / / |  temporary storage
D | / / / / |
```

Notes to Figure 9b

[1] The master routine starts in noninterpretive mode; the first two instructions call FLOATCODE, so the following lines are obeyed interpretively.

[2] It was possible to set up floating-point programme constants, but much easier to read them in from a data tape. The "instruction" ///V is an interpretive one to call the read vector routine; FLOATCODE looked after the oetails of linkage.

[3] Here we leave FLOATCODE so that ordinary instructions can be used for organizational parts of the programme and nonfloating operations.

[4] Interpretive instructions could use B-lines in the normal way. The notation F(A) means the (pseudo) floating-point accumulator.

[5] The auxiliary sub-routine is written entirely in interpretive instructions. Routines could use normal or interpretive instructions, or a mixture.

(*b*) Programme—master routine

Figure 9 continues overleaf.

Figure 9 continued.

```
   Column /
 / | @C /  Y ]
 E | @C /  W ] copy t into @C
 @ | @/ /  M   F'(A) = |t|
 A | : / / V   F'(A) = √|t| using FC:SOUAREROOT
 : | : C / Y   save in :C
 S | @C /  G ]
 I | @C /  O ]
 U | @C /  Y ]
 ½ | @C /  O ] F'(A) = 5t³
 D | @C /  Y ]
 R | : F / G ]
 J | @C /  O ]
 N | : C / W   F'(A) = √|t| + 5t³
 F | / U / £   closure
```

(*c*) Programme—auxiliary routine

```
+10.    +4.4721   3195   -02
+9.     +3.1622   7815   -02
+8.     +9.9899   9998   +02
+7.     -4.4855   8620   +01
+6.     +4.7754   1374   +01
+5.     +9.9899   9998   +02
+4.     +6.2351   5750   +01
+3.     +9.9899   9998   +02
+2.     -1.0775   5050   +03
+1.     +9.9899   9998   +02
+0.     +1.8099   7447   +01
```

(*d*) Printed output

the 1950s and early 1960s. Let us illustrate the technical aspects of AUTOCODE by showing how TPK would have been coded.

4.2.1 Technical details. AUTOCODE was intimately tied to the normal Scheme B routine changing sequence and the sub-routine library, so the problem would be broken into routines in the usual way. Figure 10*a* shows the routines for the problem; the library routines are taken from the normal Scheme B library. AUTOCODE allowed routines to be numbered 1, 2, 3, etc., according to their position in a standard Scheme B directory, which would have to be prepared by the programmer. AUTOCODE did not address the problem of scaling so we have to do the usual scaling calculation; this is shown in Figure 10*b*. Figure 10*c* shows the coding for TPK; several footnotes have been added to explain some of the more obscure points in using AUTOCODE, but even a cursory study of Figure 10 will convey the flavor of AUTOCODE.

Perhaps the feature of AUTOCODE that strikes one first is how extremely machine dependent it was. No attempt whatever was made to conceal the machine architecture. In many respects AUTOCODE had more in common with a present-day high-level assembler (for example, see Wirth 1968) than with a programming language. Certainly it did nothing to make programming easier for a novice, who would need to be very familiar with machine code programming to use AUTOCODE in the first place. Glennie claimed (1952*a*) that the system enabled an experienced programmer to write programmes much more quickly, with fewer errors, and that the resulting inefficiency was no more than about 10%.

The language itself has, for its period, some very attractive features. The assignment statement, for example, shows a finely judged balance between object code efficiency on one hand and programmer convenience on the other. The LOOP-REPEAT statement shows equally sound judgment. (Incidentally, in the first draft of the user's manual, this was SUM-REPEAT, indicating that a process of refinement and generalization was going on.) Glennie (1959, p. 29) has described the business of designing a language:

> In my experience, notations for automatic codes are constructed hy a process of introspection on the part of the inventor, who writes down possible notations and then tries to analyse how he himself recognizes them for what they are and which are the significant words. He seeks to put himself in the position of a teacher teaching a very backward child how to read.

AUTOCODE programmes were blind punched using the normal keyboard perforators. Much of the written form of the autocoded programme was notional rather than actual. For example, there was no distinction between upper- and lowercase letters, subscripts were not dropped to a lower line, and there were no layout characters. It was not possible to list a programme.

The translator occupied 10 pages of code and contained some 750 instructions: of these about 140 instructions were used for the translation of arithmetic expressions (Glennie 1952*b*). AUTOCODE did not produce diagnostics as such but it

1. INPUT/A	input vector
2. DECOUTPUT/A	print accumulator as ±M·L
3. SOUAREROOT	
4. AUXILIARY	
5. MASTER	

(*a*) List of routines

We have to calculate for each element t of the vector

$$y = \sqrt{|t|} + 5t^3 \qquad (1)$$

Assuming that t is less than about 10 in magnitude we can rewrite equation 1 as

$$y' = 2^{-13}\sqrt{2\,|t'|} + 0{\cdot}5t'^3 \qquad (2)$$

where $y' = 10^{-7}y$ and $t' = 10^{-2}t$. Now all the numbers handled are less than a half in magnitude. Notice that scaling has been done in powers of 10 rather than in powers of 2: this is because AUTOCODE allowed decimal constants to be used (so perhaps there was at least a little help with the scaling problem).

(*b*) Scaling calculation

```
          a@/½· b@VA  c@MA  d@GA  e@OA  f@Z:  y@PA[1]
          FRACTIONS +49999 → b +00004 → d +0000999 → e[2]
          f →q[3]
          SUBROUTINE 1
          LOOP 10n[4]
          n → c[5]
          +bc SUBROUTINE 2
          +aₙ SUBROUTINE 4 → y[6]
          +y−d TESTA X[7]
          +y SUBROUTINE 2 CONTROL Y[8]
ENTRY X   +e SUBROUTINE 2
ENTRY Y   REPEAT n
ENTRY Z   CONTROL Z[9]
          WRITE 5 START 5[10]
```

(*c*) Programme—master routine

```
          b@VA  c@MA  d@GA  e@OA  t@/C[1]
          FRACTIONS +49999 → b +000001 → e
          → t[2]
          +t TESTA X
          −t
ENTRY X   SUBROUTINE 3 → c[3]
          +tt → d +td → d +bd → d[4]
          +ec+d[5]
          CLOSE[6]
          WRITE 4
```

(*c*) Programme—auxiliary sub-routine

Notes to master routine

[1] Fractional scalars and vectors are identified by a single letter. Variables have to be allocated actual storage locations; thus b is at line VA, c is at line MA, and so on. The vector a is placed in systematic work space in column ½: The variable f is assigned to line Z: of PERM, which contains the value / ½// we will need later; this has to be done in this way because it was not possible to set up integer constants.

[2] The FRACTIONS statement fills variables with initial values, which must be in the range $-\frac{1}{2}\cdot \leq x < \frac{1}{2}$: Thus 0.49999 in b, 0.00004 in d, and so on (we really wanted 0.5 in b but this would be out of range so 0.49999 is used).

[3] Six integer variables were denoted by k, l, n, o, q, and r, corresponding to six of the B-lines, which were used for counting and array subscripts. This statement sets q (B_6) to the value in f (/ ½//) which in fact is a pointer to the vector stored in column ½: Next, SUBROUTINE 1 is called to input a vector using the parameter B_6 to the point to the vector.

[4] The statement pair LOOP 10n . . . REPEAT n causes the intervening statements to be obeyed repeatedly. The index n is initialized to 20 (i.e., twice 10) and decremented by 2 each time around the loop until it gets to zero.

[5] The effect of this line and the next is to multiply the index by a half and print it using sub-routine 2.

[6] We now load a_n into the accumulator and call the auxiliary sub-routine. The accumulator is used for the argument and result. The result is stored in y when we return from the sub-routine.

[7] TESTA X transfers to the point labeled ENTRY X if the accumulator ≥ 0.

[8] y is printed using sub-routine 2. CONTROL Y is an unconditional transfer of control.

[9] The programme terminates on a "dynamic stop."

[10] This line is a directive to write the master routine to the magnetic store as routine number 5, and then enter it.

Notes to auxiliary sub-routine

[1] Variables and initial values are assigned in the usual way in the first two lines. Note that t has been put in column C; it is not stored locally like the other variables because when the square-root routine is brought down later it would be destroyed.

[2] The argument is transferred to AUXILIARY via the accumulator, so we save it in t and set the accumulator to |t|.

[3] Sub-routine 3 forms the square root of the accumulator, which we save in c.

[4] These statements form $0{\cdot}5t^3$. It was not possible to have terms containing more than two operands; this was a limitation imposed for efficiency reasons to make the best use of the Mark I instruction set.

[5] This leaves $10^{-6}\sqrt{|t|} + 0{\cdot}5t^3$ in the accumulator.

[6] CLOSE returns to the master routine.

Figure 10. TPK in Glennie's AUTOCODE.

would come to a stop if it detected a syntax error. The point where the programme tape halted in the reader then gave the location of the faulty statement.

AUTOCODE was very fast and, programme for programme, was about as fast as the normal input routine. This was partly because autocoded programmes used many fewer symbols. For this reason, programmes were never stored in binary but were translated each time they were used. Probably TPK would have taken well under a minute to translate. (This contrasts startlingly with contemporary American automatic coding systems, such as the UNIVAC A-series compilers, which would have taken much longer.[7] (Incidentally, all the Manchester University machines have tended to use fast load-and-go compilers, and such things as linkage editors and object code are not part of the Manchester tradition.)

4.2.2 Historical details. Until Knuth and Trabb Pardo's recent paper (1976), very little was known of Glennie's work. Glennie himself never published an account of AUTOCODE, although a few brief comments were published to the effect that there was an automatic coding system in existence.[8] Glennie never published partly because it was not usual for scientists in the employ of the Ministry of Supply to publish their work—and Glennie was engaged in highly classified nuclear weapons research. In any case, not being in the academic community, Glennie had little need for publications to make his reputation. Glennie also recalls that he found it difficult to compose a paper in the accepted scientific style that would have appealed to a learned journal.

Glennie's recollection is that the idea of automatic coding was very much in the air in the early 1950s. It was well known that Aiken had constructed a special-purpose coding machine for the Harvard Mark III (Harvard 1952). It was equally well known that there was "no need to build a special machine for coding, since the computer itself being general purpose should be used for coding" (Glennie 1952*b*).

AUTOCODE was designed for the specific purpose of reducing the drudgery of coding and hence the incidence of programming errors. Of all the early Mark I users, Glennie probably had by far the biggest problems to solve, so that for him the burden of programming in teleprint code was most acute. In the atmosphere of a university laboratory, one was prepared to try radical approaches and Glennie undertook the development of AUTOCODE in the spirit of an interesting sideline to his proper work.

AUTOCODE was written during about three months of part-time activity in the summer of 1952; in September it was described by Strachey (1952) as "just coming into use." Unfortunately, Glennie destroyed the original tapes and listings of AUTOCODE many years ago. However, he did prepare two drafts of a user's manual for AUTOCODE that give a detailed account of its use (Glennie 1952*a*).

Shortly after AUTOCODE came into use, Glennie was invited hy M. V. Wilkes to speak on the subject at one of the Cambridge seminars. Fortunately, Glennie's employers had a strict rule that staff members intending to publish or lecture externally first had to submit the proposed dissertation (in triplicate) for security clearance. Glennie prepared typed lecture notes (Glennie 1952*b*) for the seminar that contain a detailed account of the internal structure of the translator and the nature of the code generated. Glennie gave his seminar in February 1953; by and large it was well received, although its impact in making people do things differently was very small.

In the early 1950s the use of automatic coding was very far from being universally accepted. For example, Gill was rather disparaging about the value of Glennie's work in the discussion following the Cambridge seminar and he repeated these remarks almost word for word at the NPL symposium in March 1953. Glennie (1977) recalls telling Turing about AUTOCODE. At first Turing was quite interested, until he realized that Glennie had achieved formula translation rather than formula manipulation; "then the shutters came down, it just wasn't up to his intellectual standard." On the other hand Strachey (1952) was much more enthusiastic and announced it as "a first and very important step." (Incidentally, the Strachey papers contain notes for an automatic coding scheme entitled "Generalised Programming" with some notational similarities to Glennie's AUTOCODE. These notes are undated but would seem to be roughly contemporary with AUTOCODE.)

AUTOCODE was in fact only used by Glennie himself. Probably he could have done much more to sell AUTOCODE to other users but his position as an external user no doubt made him reti-

[7] See, for example, UNIVAC (1956), where compilation times of about 10 minutes are typical.

[8] See, for example, Strachey (1952), Bennett and Glennie (1953), Mutch and Gill (1953).

cent. Glennie did use AUTOCODE for several problems of his own, so the development was not without some benefit. The problems solved each had about 100–300 statements. Soon after this, Glennie became involved full time with a much larger problem for which AUTOCODE was inappropriate, and so it died a natural death.

Glennie (1965–1967) has described AUTOCODE as "a successful but premature experiment." The reasons that AUTOCODE had so little impact bear some examination for the insight they give into what factors made for a successful automatic programming system in the early 1950s. The most serious shortcoming of AUTOCODE was that it did not address the problems of scaling or of the two-level store. There was little incentive to use AUTOCODE when the programmer was still left to deal with scaling, which was by far the most difficult part of programming. Furthermore, AUTOCODE was incapable of tackling problems that required use of the magnetic store for holding arrays of data (such as matrix problems). An important feature of any automatic coding system was the extent to which it could be readily used by inexperienced users; AUTOCODE signally failed in this respect. Perhaps the final lesson is that to be a success, an autocode needs committed full-time programming support; Glennie could not provide this.

4.3 The Mark I Autocode

In 1954, Brooker introduced a system of automatic coding specifically designed to enable nonexpert users to run small problems. The system, at first called "the simple machine" and later the "Mark I Autocode," attacked simultaneously all three problems facing the Mark I programmer: scaling, the two-level store, and the language difficulty. Let us first see how the TPK programme would have been coded in the Mark I Autocode.

4.3.1 Technical details. The Mark I Autocode was a very polished system. The TPK programme shown in Figure 11*a*[9] is all that a programmer would need to prepare; there were no complications such as selecting library routines or punching directories. Autocode programmes were punched on teleprinters especially acquired for the Autocode service, and a listing of the programme was automatically produced.

The Autocode TPK programme has been run on the Mark I simulator and the printed output is shown in Figure 11*b*. It was not possible to lay out the results in the form of a table because the printing of a number is invariably accompanied by a new line. These rather rudimentary output facilities were probably not a serious handicap for the kinds of problem run using Autocode, which tended to be small one-off jobs.

The following description of Autocode is based on a written account given in the supplement to the third handbook (HB3S), which includes a complete listing of Autocode. In addition, Brooker's original coding sheets for Autocode are still in existence, and these have been used to resolve certain ambiguities. Figure 11*c* shows the store layout of an Autocoded programme. Autocode was based on the standard Scheme C floating-point routines, which were kept permanently in the electronic store. The programme itself was translated into a chain of routines that were stored on consecutive tracks of the drum. During execution, the programme was brought down two pages at a time, transfers of control between pages being completely taken care of by Autocode.

The storage management of floating-point variables was perhaps the most significant feature of Autocode. The several thousand floating-point variables were stored on consecutive tracks of the drum. Whenever a variable was accessed, the MAG. LOCATION routine computed the drum address of the variable and brought the relevant track into the electronic store. Appropriate measures were taken to avoid bringing a page down again once it was already in the electronic store. Similarly, pages that had not been altered were not written back to the drum. Brooker called this concept the "one-level" store. Some of the basic features of a page-changing algorithm for virtual storage can be seen in embryo.

The Autocode translator took about 2 minutes to load from paper tape, but this only had to be done once for a batch of programmes. The compiling speed was about 2 seconds per instruction, and the execution speed about 6 instructions per second. The TPK programme would have taken about 100 seconds: 40 seconds to translate and 60 seconds to execute. (Execution would, of course, have taken longer if the variables had not all been stored in the same page.) In general programmes ran between five and fifty times slower than equivalent machine code programmes. The code generated by Autocode was very crude, mainly

[9] This TPK programme is essentially the same as in Knuth and Trabb Pardo (1976, p. 64) with the correction of a small error.

```
     Programme            Notes

1        n1 = 1                                    ┐
3       vn1 = I           inputs a_i [1]           │ cycle 11 times [2]
         n1 = n1 + 1                               │
   j3,   11 ≥ n1          tests for last cycle     ┘
         n1 = 11                                   ┐
2     *  n2 = n1 − 1      prints i [3]             │
        v12 = F6(vn1)  ┐                     ┐     │
        v12 = F1(v12)  ┘  v12 = √|a_i| [4]   │     │
        v13 = 5⊗vn1    ┐                     │ [5] │
        v13 = vn1⊗v13  │  v13 = 5a_i^3 [6]   │     │
        v13 = vn1⊗v13  ┘                     ┘     │
        v12 = v12 + v13   y = √|a_i| + 5a_i^3      │ cycle 11 times [2]
   j4,  v12 > 400                                  │
      * v12 = v12         prints y                 │
         j5                                        │
4     * v12 = 999         prints 999               │
5        n1 = n1 − 1                               ┘
   j2,   n1 > 0           tests for last cycle
         H                halts
        (j1)              starts programme [7]
```

(*a*) Programme

```
+10.
+.04472
+9.
+.03162
+8.
+999.
+7.
−44.85586
+6.
+47.75413
+5.
+999.
+4.
+62.35157
+3.
+999.
+2.
−1077.55051
+1.
+999.
+.
+18.09974
```

(*b*) Printed output

*Notes to Figure 11*a

Floating-point variables were denoted v1, v2, v3, . . . , up to about v5000; these could be used as scalars or vectors. Integer variables, "indices," were denoted n1, n2, . . . , n18 and were used for counting operations and subscripts. For the TPK problem, the vector is stored in v1 to v11.

[1] I on the right-hand side of an "equation" caused the next value to be read from the number tape.

[2] Programme loops were spelled out in full, there being no specific loop control statement. 'j3' means jump to the statement labeled 3.

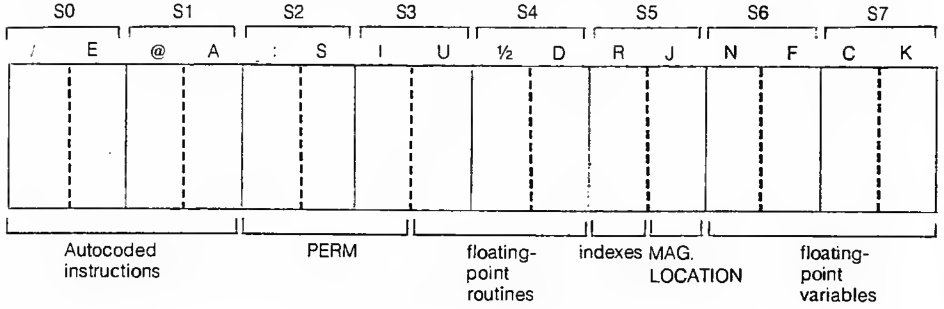
[3] An asterisk (*) before an equation caused the result to be printed. This was very useful for debugging; once a programme was working, the *'s could be overpunched with an erase symbol.

[4] F6 and F1 are the absolute value and square root functions, respectively. Function calls could not be nested.

[5] There was no sub-routine facility, so the coding for a procedure has to be given in-line.

[6] An arithmetic expression could be no more complicated than a single binary operation on two operands. Hence $5t^3$ has to be calculated using several statements.

[7] Instructions in parentheses were obeyed immediately as an "interlude." This sequence is used to start the programme by jumping to label 1.



(*c*) Storage organization

Figure 11. TPK in the Mark I Autocode.

because there was nothing to be gained by producing efficient code. Even though only about five Autocode instructions were accommodated on each two-page track, there was still adequate room on the magnetic store for programmes of several hundred instructions. Further, efficiently generated code could do nothing to overcome the built-in inefficiency of floating point and the one-level store.

Autocode gave no diagnostics at all and tended to digest anything it was given—erroneous programmes produced erroneous results. For well-formed programmes Autocode generally behaved satisfactorily, however. Just one serious error has been found in the listings of Autocode: integer division is not performed. This is quite clearly a simple coding error, and presumably the surviving documentation does not reflect the fully debugged compiler, for such a gross error could not have lain dormant for very long.

Autocode was a very big programme for its day. The code can be broken down as follows:

	Pages	Approx. no. of instructions
Translator	36	1800
Floating-point routines	6	300
Mathematical functions, I/O, Runge-Kutta, etc.	16	800
Complex routines	14	700
Total	72	3600

Autocode included facilities for the solution of differential equations by Runge-Kutta and calculations involving complex variables—reflecting the needs of the users.

4.3.2 Historical details. A first version of the Mark I Autocode was developed by Brooker early in 1954, and a paper describing the system was submitted in March (Brooker 1955). Livesley and Popplewell worked on the first application for an outside user in late 1954 (this was a small set of one-off calculations for a local mechanical engineering firm). In the summer of 1955, Brooker rewrote the entire Autocode system, and this version is described and listed in the supplement to the second handbook. Brooker's original programme manuscripts are dated from May to August 1955. As well as some slight syntactical changes, the second version of Autocode included facilities for complex variables.

Brooker was well aware of Glennie's earlier AUTOCODE, of course, although he had not himself used it. But in fact the systems had no more in common than the name *autocode* and were completely separate developments. Interestingly, Brooker was reluctant to call his system "autocode" and first called it "the simple machine." In a contemporary letter to Strachey (late 1954) Brooker wrote: "Incidentally I am coining the phrase Unicode for my autocode—mainly to avoid comparison with Glennie's." But "autocode" rapidly came into the language as the British equivalent of "programming language," and Brooker doubtless acceded to fashion.

With hindsight, we can see that the most significant innovation of the Mark I Autocode was the concept of the one-level store. Brooker recalls that this was a rare moment of inspiration; it just suddenly occurred to him that since there was already the inefficiency due to floating point, it would be worth the additional inefficiency of having a one-level store. The one-level store as implemented in the Mark I Autocode was one of the seminal ideas that achieved final expression in the virtual store of the Atlas.

Considered as a programming language, rather than as a simplified means of using the Mark I, Brooker's Autocode is less impressive. Certainly, the control structures and arithmetic expressions are not as well developed as in Glennie's AUTOCODE. To be fair, Brooker did not consider the design of a sophisticated programming language to be part of his brief; his aim was to devise a method of simplifying programming, particularly for inexperienced users. In this Brooker succeeded admirably; as Gill put it (Lunt 1957):

> This work calls for a tremendous amount of experience and sympathy with the programmer who will eventually use the scheme and a number of attempts have failed because one or other factor was not taken into account. Dr. Brooker seems to be more capable than many people I know of choosing just the right combination of features to be included in a scheme of this kind.

Gill in fact gave Brooker considerable help with promoting Autocode.

The language of the Mark I Autocode was so simple that Brooker (1958, p. 16) subsequently described it as "amounting to what is nothing more than a two- or three-address instruction code." The simplicity of the language was instrumental in getting it accepted by new users. (A very similar aim was much later embodied in BASIC.) The complete programmer's description of Autocode occupied just two sides of foolscap, with a third side to describe an example. Brooker spent a great deal of time polishing this "standard account." A surviving description of Autocode entitled "The Standard Account of the Simple Machine" is dated June 1955.[10] With supplementary sheets describing the use of the Runge-Kutta function and complex variables, the documentation grew to a total of nine sides. Autocode was so easy to teach and to learn that new users could be trained in a matter of hours.

With the arrival of Autocode a small computing service was established, and a machine operator and a keypunch operator were hired at the Computing Machine Laboratory. The usage started at about one hour per day in 1954 and gradually built up to perhaps three or four hours per day. Autocode produced a breakthrough in programming by reducing programming and testing time by up to a factor of one hundred (Richards 1979). A postal Autocode service was set up wherein users mailed their programme manuscripts, which were then punched and run, with a usual turnaround time of less than a week. Following its completion in 1955, some 12 hours of machine time per week was used for the Autocode postal service alone (Brooker 1958, p. 16). The service was enthusiastically used by the Central Instrument Laboratory of Imperial Chemical Industries where G. E. Thomas now headed the computer section, assisted by B. Richards, both formerly associated with the Mark I at Manchester. At ICI, courses were given in Autocode and scientists developed their own programmes, typically for problems in chemical plant design and statistical calculations (Thomas 1978). Other users of the postal service included the Theoretical Physics Branch of the Atomic Energy Research Establishment at Harwell.

Autocode was a tremendous success, particularly with engineering users for whom time was money. Autocode enabled them to get programmes working with unprecedented speed. Brooker recalls being buttonholed one day by a delighted user for whom Autocode had just saved several tens of thousands of pounds on a civil engineering contract. At a March 1957 meeting of the Society of Instrument Technology, S. T. Lunt, from ICI's Central Instrument Laboratory, gave an account of the economic and other benefits of using Autocode in chemical engineering design (Lunt 1957). The meeting was a great success and did a lot to promote Autocode. Such was the enthusiasm at the meeting that Brooker found himself the man of the moment and was invited to dine with the inner circle of the society. Automatic coding had arrived.

5 Assessment

The Mark I was clearly a difficult machine to programme, but this is a judgment made in retrospect; most contemporary users had nothing with which to compare it and consequently had little complaint.

A notable feature of life in the Computing Machine Laboratory was what might be called lack of management control—or to put it another way "all chiefs and no Indians" (Glennie 1979). But this atmosphere gave a freedom to experiment in automatic coding systems that might not have otherwise developed. The Manchester autocodes evolved to simplify programming; indeed so successful was Brooker's Autocode in this respect that the Mark I became possibly the easiest machine to programme in Britain.

5.1 The Early Programming System

Turing's programming scheme for the prototype Mark I was a prosaic piece of work. Given the exigencies of developing an input routine very quickly for a machine that was never destined to be a permanent fixture, this was understandable. Perpetuating an essentially similar scheme on the production Mark I was unfortunate, however. The arguments for programming directly in teleprinter code, largely based on correspondence between the written form and the monitor tube display, were insubstantial—and never came about at Cambridge, for example. It seems that Turing did not recognize the importance of clear notation in programming; he himself had no difficulty and he did not expect others to. A remark by Wilkes (1968, p. 4) throws some light on Turing's attitude:

> He had a very nimble brain himself and saw no need to make concessions to those less well-endowed. I remember that he had decided—presumably because someone had shown him a train of pulses on an oscilloscope—that the proper way to write binary numbers was backwards, with the least significant digit on the left. He would, on occasion, carry this over into decimal notation. I well remember that once, during a lecture, when he was muliplying some decimal numbers together on the blackboard to illustrate a point about checking a program, we were all unable to follow his working until we realized that he had written the numbers backwards. I do not think that he was being funny, or trying to score off us; it was simply that he could not appreciate that a trivial matter of that kind could affect anybody's understanding one way or the other.

[10] The same standard account is given as an appendix to Brooker (1955, 1956).

Even if the suggestion for a more suitable programming notation had been put to Turing, one feels he would have been unlikely to respond to it. It is clear that elegance or cleverness in programming was not a subject that interested Turing. The only exception to this was Turing's "formal mode," but even here the actual implementation was awkward and unpolished and no doubt contributed to its lack of use. A recollection of Tootill's (1978) sums up Turing's attitude to programming:

> Turing looked on programming as boring routine work—he seemed in fact to regard tidying up a working program as a waste of time, like, for example, turning all the pennies in your pocket the same way up.

One of the hopes for Brooker's appointment to the Computing Machine Laboratory, in October 1951, was that he would improve matters for the users. The subsequent programming systems, Schemes B and C, were attempts to streamline the input process. That it was not possible to break with the teleprinter notation is an interesting example of what might be called the "inertia factor" that has dogged many software projects: namely, early design decisions, both good and bad, tend to be perpetuated. (An outstanding example of this is the FORTRAN DO-loop—not to say the entire FORTRAN language.) It seems to be an almost superhuman endeavor to make a change *and* persuade others to follow.

5.2 The Autocodes

The primitive state of the programming system did have a valuable side effect in stimulating the development of automatic coding systems. This was quite different from the situation at Cambridge where the programming system developed by Wilkes and Wheeler was so satisfactory that much of the incentive for making improvements was removed.

Glennie's AUTOCODE was an outstanding innovation. Its failure to catch on had two main causes: first, it did not overcome the worst features of programming the Mark I (scaling and storage management), and second, it did not attract the support of the laboratory (perhaps a natural reticence on the part of Glennie had some part to play here).

The real success story of the programming effort was Brooker's Mark I Autocode. The success of the Mark I Autocode (especially when contrasted with Glennie's) can be attributed to several factors:

1. *Ease of use.* Autocode was easy to learn and use; it attracted a body of new users rather than trying to win over existing programmers.
2. *Fulfilled a need.* Autocode offered significant facilities not otherwise readily available (floating point, one-level storage, simplified input/output).
3. *Effective selling.* Autocode was promoted by a series of programming courses, an exceptionally short and comprehensible manual, and a postal service for remote users.
4. *Committed support.* Autocode had the full support and enthusiasm of the laboratory in terms of maintenance, documentation, programming advice, and consultancy.

The above would make an effective checklist today for the launch of any new software product.

Brooker's Mark I Autocode was probably the most significant programming innovation of the mid-1950s in Britain: it inspired several imitations at the time when American directions were not widely reported in Britain. For example, G. E. Felton, programming manager for the successful Ferranti Pegasus, initiated the Pegasus autocode, which was released in the summer of 1958 (Felton 1978); from this grew the autocode for the Ferranti Sirius computer shortly after. Also in the late 1950s autocodes for the Elliot 802 and 803 were produced. All these systems had an extremely strong family resemblence to Brooker's progenitor.[11] Brooker's Mark I Autocode, and his subsequent Mercury Autocode of 1958, was a dominant influence on British autocodes until well into the 1960s. It was only in the middle and late 1960s that ALGOL 60 and FORTRAN became a significant force for British computers.

5.3 Conclusions

Apart from the one bright spot of Brooker's Mark I Autocode, it is clear that the place of the Manchester Mark I in the history of computer development is secured far more by its hardware than by its software. The Mark I had three brilliant hardware innovations: the first operational CRT memory, the index register; and (albeit

[11] Burnett-Hall et al. (1964) describes several early British autocodes of this type.

somewhat unwittingly) embryonic virtual memory hardware. The early software failed in any way to match these triumphs. For example. W. G. Welchman, who had contemporary experience on both sides of the Atlantic, recalled (1979):

> When I returned to England in early 1954 . . . I was appalled by the contrast between what was going on in America in applications and programming and what I found in Manchester. . . . Ferranti were already offering their Mark I computer in America in competition with the ERA 1101. But the British engineers were very poorly supported by programmers [and] applications research.

Significantly, the Mark I found no takers in North America. But Welchman's judgment is perhaps harsh; software generally played second fiddle to the hardware in early computers. Also, when it arrived in March 1954. Brooker's Mark I Autocode would have stood comparison with anything developed in America; it is unfortunate that Ferranti failed to exploit it on Mark I* computers.

Finally, it is appropriate to say a word about Turing's contribution to the work of the laboratory. which thus far has appeared to be somewhat negative. Turing certainly made some positive contributions. For example. he was an accomplished mathematician and often helped others in matters of analysis and numerical methods. On his own account, Turing was an indefatigable programmer: he was probably the heaviest user of the Mark I (two whole nights every week) (Hodges 1979). It was during this period that he did much of his work on morphogenesis (Turing 1952), which his death in June 1954 cut short. In those with whom he worked. Turing inspired awe and often affection. As Glennie recalled recently (1979):

> Turing's manner of thought was so different from most other people's that his ideas both irritated and invigorated. You had to be mentally prepared for all sorts of unusual ideas where Turing was.

Acknowledgments

It is a pleasure to acknowledge the help and time freely given to me by several of the pioneers at Manchester, in particular T. Kilburn, R. A. Brooker. A. E. Glennie, C. M. Williams (née Popplewell), and D. G. Prinz.

The following people read and criticized an earlier draft of this report and/or answered specific questions: G. E. Felton, B. A. Galler, A. Hodges, N. E. Hoskin. S. H. Lavington, R. K. Livesley, B. Richards. G. E. Thomas, G. C. Tootill, and W. G. Welchman. F. H. Sumner acted as principal referee for this paper, and several anonymous reviewers made suggestions that greatly improved its readability. Any remaining errors or shortcomings are the responsibility of the author, of course.

I would like to thank S. H. Lavington who helped me on numerous occasions in preparing this paper and also supplied the photographs. I am grateful to D. E. Knuth for permission to use the TPK algorithm. Mrs. J. B. Alton, of the Contemporary Scientific Archives Centre, and Mrs. B. Halpern allowed me access to the Strachey papers. My thanks to June Miller, who punched several thousand Mark I instructions with great accuracy, and Janet Johnson and Audrey Davison. who typed the manuscript. Finally, my thanks to B. Randell and A. P. Steward, who guided me.

REFERENCES

AIEE-IRE. 1951. "Review of Electronic Digital Computers." Joint AIEE-IRE Computer Conference. Philadelphia.

Bates, M. A. 1950. *On the Mechanical Solution of a Problem in Church's Lambda Calculus.* M.Sc. Thesis, Manchester University.

Bennett. J. M.. D. G. Prinz, and M. L. Woods. 1952. Interpretative sub-routines. *Proc. ACM Nat. Conf.*, Toronto, 81–87.

Bennett. J. M.. and A. E. Glennie. 1953. *Programming for High-Speed Digital Calculating Machines.* Bowden, pp. 101–113.

Booth, A. D. 1975. Computers in the University of London. 1945–1962. *Radio and Elect. Eng. 45,* 7, 341–345.

Bowden. B. V. (ed.). 1953. *Faster than Thought.* London: Pitman.

Brooker, R. A. 1955. An attempt to simplify coding for the Manchester electronic computer. *Brit. J. Appl. Physics 6*, 307–311.

Brooker. R. A. 1956. The programming strategy used with the Manchester University Mark I computer. *IEE*, 151–157.

Brooker. R. A. 1958. The Autocode programs developed for the Manchester University Computers. *Computer J. 1,* 15–21.

Brooker, R. A. October 21, 1978. Conversation with the author.

Brooker, R. A., I. R. MacCallum, D. Morris, and J. S. Rohl. 1964. The compiler compiler. *Ann. Rev. in Automatic Programming 4*, 229–275.

Brooker. R. A., and D. J. Wheeler. 1953. "Floating Operations on the EDSAC." *Mathematical Tables and Other Aids to Computation 7*, 37–47.

Burnett-Hall, D. G., L. A. G. Dresel, and P. A. Samet. 1964. *Computer Programming and Autocodes.* London: English Universities Press.

Cambridge University. 1949. "Report of a Conference on High Speed Automatic Calculating Machines 22–25 June 1949." University Mathematical Laboratory.

Campbell-Kelly, M. 1980. Programming the EDSAC: early programming activity at the University of Cambridge. *Annals of the History of Computing*, Vol. 2, No. 1, 7–36.

Carpenter, B. E., and R. W. Doran. 1977. The other Turing machine. *Computer J. 20*, 3, 269–279.

Computer Weekly. April 10, 1969. In the beginning: tracing the steps of the Ferranti Mark 1, 12–13.

Felton, G. E. May 1, 1979. Letter to the author.

Gill, S. 1951. The diagnosis of mistakes in programmes on the EDSAC. *Proc. Roy. Soc. (A) 206*, 538–554.

Glennie, A. E. c. 1952*a*. "Automatic Coding." Unpublished notes (two versions), undated (probably 1952).

Glennie, A. E. 1952*b*. "The Automatic Coding of an Electronic Computer." Unpublished lecture notes.

Glennie, A. E. 1959. Automatic programming: a simplified technique. *Data Processing*. Illiffe. pp. 26–29.

Glennie, A. E. 1965–1967. Letters to D. E. Knuth.

Glennie, A. E. July 26, 1977. Conversation with the author.

Glennie, A. E. November 17, 1978. Letter to the author.

Gotlieb, C. C. 1954. Running a computer efficiently. *JACM 1*, 124–127.

Harvard University. 1952. Description of a magnetic drum calculator. *Annals of the Computation Laboratory of Harvard University 25*.

HB1. "Programmers' Handbook for Manchester Electronic Computer Mark II." (A. M. Turing, ed.). March (?) 1951; errata March 13, 1951.

HB2. "Programmers' Handbook (2nd Edition) for the Manchester Electronic Computer Mark II." (R. A. Brooker, ed.). August 1952.

HB2S. "Supplement to the Programmers' Handbook (2nd Edition) for the Manchester Electronic Computer Mark II." (R. A. Brooker, ed.). Undated, probably late 1952–early 1953.

HB3. "Programmers' Handbook (3rd Edition) for the Manchester Electronic Computer Mark II." (R. A. Brooker, ed.). September 1953.

HB3S. "Manchester Electronic Computer Mark I. Supplement to the Programmers' Handbook (3rd Edition)." (R. A. Brooker, ed.). January 1956.

Hodges, A. November 17, 1978. Letter to the author.

IEE. 1956. Convention on digital-computer techniques. *Proc. IEE 103*, Part B Supplement.

Kilburn, T. March 8, 1979. Conversation with the author.

Kilburn, T., R. B. Payne, and D. J. Howarth. 1961. The Atlas supervisor. *Proc. EJCC, AFIPS 20*, 279–294.

Kilburn, T., and L. S. Piggot. 1978. Frederick Calland Williams 1911–1977. *Biographical Memoirs of Fellows of the Royal Society 24*, 583–604.

Knuth, D. E. and L. Trabb Pardo. 1976. *The Early Development of Programming Languages*. STAN-CS-76-562, Computer Science Department, Stanford University.

Lavington, S. H. 1975. *A History of Manchester Computers*. Manchester: NCC Publications.

Lavington, S. H. 1978. The Manchester Mark I and Atlas: a historical perspective. *CACM 21*, 1, 4–12.

Livesley, R. K. February 26, 1979. Letter to the author.

Lunt, S. T. 1957. Process development and plant design: the role of instrumentation with particular reference to the applications of computers. *Trans. Soc. Instr. Technology 9*, 87–103.

Manchester University. 1951. "Manchester University Computer Inaugural Conference."

Michie, D. August 29, 1978. Letter to the author.

Mutch, E. N., and S. Gill, 1953. Conversion Routines. (NPL 1953), pp. 74–80.

Newman, M. H. A. June 14, 1949*a*. Letter to the *Times* (London).

Newman, M. H. A. 1949*b*. "Some Routines Involving Large Integers." Cambridge University, pp. 69–70.

Newman, M. H. A. 1955. Alan Mathison Turing 1912–1954. *Biographical Memoirs of Fellows of the Royal Society 4*, 253–263.

NPL. 1953. "Automatic Digital Computation." Proceedings of a symposium held at the National Physical Laboratory, March 25–28, 1953. HMSO 1954.

Popplewell, C. M. 1969*a*. A user's view of the 1949 Manchester machine. Unpublished notes. 2 pp.

Popplewell, C. M. 1969*b*. Programming for the Ferranti Mk I. Unpublished notes, 2 pp.

Popplewell, C. M. April 14, 1978. Conversauon with the author.

Prinz, D. G. 1952. "Introduction to Programming on the Manchester Electronic Digital Computer." Ferranti.

Prinz, D. G. May 31, 1978*a*. Conversation with the author.

Prinz, D. G. November 20, 1978*b*. Letter to the author.

Randell, B. 1976. *The COLOSSUS*. University of Newcastle-upon-Tyne Computing Laboratory, Technical Report Series, 90.

Richards, B. October 30, 1979. Letter to the author.

Strachey, C. 1952. Logical or non-mathematical programming. *Proc. ACM Nat. Conf.*, Toronto, pp. 46–49.

Strachey, C. 1965. A general-purpose macro generator. *Computer J. 8*, 225–241.

Strachey, C. 1966. Systems Analysis and Programming. *Information*. San Francisco: Freeman, pp. 56–75.

Thomas, G. E. May 29, 1978. Letter to the author.

Tootill, G. C. 1948–1949. Notebook.

Tootill, G. C. November 17. 1978. Letter to the author.

Turing, A. M. 1949. "Checking a Large Routine." Cambridge University, pp. 67–68.

Turing, A. M. 1951. "Local Programming Methods and Conventions." Manchester University, p. 12.

UNIVAC. c. 1956. "Typical Application of A-2 Compiler." Internal memorandum of UNIVAC Inc.

Wheeler, D. J. 1950. Programme organisation and initial orders for the EDSAC. *Proc. Roy. Soc. (A)202*, 573–589.

Wilkes, M. V. 1968. Computers then and now. (1967 ACM Turing Lecture.) *JACM 15*, 1–7.

Wilkes, M. V., D. J. Wheeler, and S. Gill, 1951, 1957. *The Preparation of Programs for an Electronic Digital Computer*. Cambridge, MA: Addison–Wesley.

Williams, F. C. 1953. *Madam*. (NPL 1953), pp. 35–38.

Williams, F. C. 1975. Early computers at Manchester University. *Radio and Elec. Engineer 45*, 7, 327–331.

Williams, F. C., and T. Kilburn, 1951. The University of Manchester Computing Machine. Manchester University, pp. 5–11. (Reprinted in AIEE-IRE 1951 and Bowden 1953.)

Williams, F. C., T. Kilburn, and G. C. Tootill. 1951. Universal high-speed digital computers: a small-scale experimental machine. *Proc. IEE 98*, Part II, 13–28.

Wirth, N. 1968. PL360, a programming language for the 360 computers. *JACM 15*, 37.

APPENDIX

The TPK Algorithm

In their report *The Early Development of Programming Languages* (1976), Knuth and Trabb Pardo argue that the best way to understand a programming language is to study specimen programs; this communicates the flavor of a language far more effectively and concisely than a lengthy programming manual. The authors introduce the TPK algorithm in their report.

The TPK algorithm is a short program that demonstrates many of the characteristic features of a program; by coding TPK in a variety of languages Trabb Pardo and Knuth have been able to contrast a number of historic programming languages in a succinct yet informative fashion.

A version of the TPK algorithm written in ALGOL 60 is given in Figure A1*a*. Figure A1*b* shows the output from the program for a given set of test data (when run on an ICL 1903A computer). The TPK algorithm demonstrates the following points: the use of variables, constants, and a vector; a program loop proceeding by positive increments and another by negative increments; accessing successive vector elements; a conditional statement; built-in functions, such as square root and absolute value; input/output procedures; a user-written procedure. The program is quite short in duration and would probably have taken between a few seconds and a couple of minutes to run on a first-generation computer, depending on how fast the computer was and how effective the language and its translator.

Of course, TPK does not actually do anything useful, but it would be difficult to devise a more illustrative program using fewer statements.

```
begin integer i; real y; real array a[0: 10];
  real procedure f(t); value t; real t;
    f := sqrt(abs(t))+5*t ↑ 3;
  for i := 0 step 1 until 10 do a[i] := read;
  for i := 10 step −1 until 0 do
  begin newline(1); print(i,2,0);
    y := f(a[i]);
    if y > 400 then print(999,4,5)
              else print(y,4,5);
  end
end of TPK algorithm;
```

(*a*) ALGOL 60 program

```
Test data:   1.5  8  −6  9.5  2.3  9.9
             2.1  −2.1  6  0.001  −0.002

Printed output:            10         0.04472
                            9         0.03162
                            8       999.00000
                            7       −44.85586
                            6        47.75414
                            5       999.00000
                            4        62.35158
                            3       999.00000
                            2     −1077.55051
                            1       999.00000
                            0        18.09974
```

(*b*) Test data and results

Figure A1. The TPK algorithm in ALGOL 60.

The Printed Papers of Charles Babbage

ALFRED W. VAN SINDEREN

This paper is intended to correct the list of printed papers of Charles Babbage (1791–1871) that Babbage printed in the back of his partial autobiography, Passages from the Life of a Philosopher *(London, 1864). That list has been reprinted in several modern publications, but no attempt has been made to make it accurate or complete until this article. In an effort to make the list more useful to the reader, notes follow many of the items to explain their significance in Babbage's life and work.*

No manuscript items are included; the paper deals only with Babbage's list of printed works, as they existed at the time of his death.

Keywords and phrases: Babbage, bibliography, publications, mathematics, Analytical Engine, Difference Engine

CR category: 1.2

Introduction

"I have frequently had applications to write my life, both from my countrymen and from foreigners," wrote Charles Babbage in *Passage from the Life of a Philosopher* (1864, p. vii); "to many of these I sent a list of my works, with the remark that they formed the best life of an author."

Indeed they do, in large measure, when accurately listed and their significance at least partially explained. Unfortunately, Babbage's lists were not accurate, nor were they supplemented with any explanation or description. This paper attempts to correct that situation and thereby assist the reader who wishes more fully to understand and appreciate this gifted and farsighted man. A few preliminary words are needed for the benefit of those who know little about Babbage, his life, and his work.

Charles Babbage, the Man

For the past 25 years or so, Charles Babbage has been known as a scientific pioneer whose work presaged the development of modern computer technology. Many words of praise were written about Babbage after his death in 1871, but modern evaluation of his work probably took hold in 1953 with the publication of B. V. Bowden's *Faster Than Thought*.[1] Calling Babbage "one of the most farsighted men of his generation," Bowden went so far as to say: "he understood clearly all the fundamental principles which are embodied in modern digital computers." Speaking of the Analytical Engine, which was Babbage's second major effort in calculating machines, Bowden described it as the "first universal digital computer, as the expression is understood today." Bowden did not attempt to prove a direct connection between the Analytical Engine and the computer technology of the 1950s, and that issue is still debated today.

Other writers have examined Babbage's machines in more detail. Philip Morrison and Emily Morrison (1961) drew heavily on Babbage's own descriptions of his work and on a book by the inventor's son, Henry P. Babbage, published in 1889. Perhaps the most thorough study of Babbage's engines is Bruce Collier's unpublished Ph.D. thesis (1971), "The Little Engines That Could've."

Characteristically, Babbage did not carry through to conclusion any of his efforts on calculating engines. He worked 10 years on his first Difference Engine, which was designed to calculate and produce mathematical tables. As discussed in Collier's dissertation, Babbage frequently enlarged his vision of what the machine could be made to do, which kept it in a constant state of redesign. There were mechanical problems stemming from the state of the art in making machinery at that time. He also encountered considerable difficulty with support from the British government because of misunderstandings that are too complex to recount here. These delays and problems allowed him time to think of a whole new concept (the Analytical Engine), and he pursued the design of this concept off and on for the rest of his life. During the late 1840s, he worked for a while on another machine he called Difference Engine No. 2. All of these efforts, including his relations with government, are discussed by the Morrisons and are summarized or glossed over lightly by several other authors.[2] Anyone interested in the whole story should con-

Author's address: 227 Church Street, New Haven, CT 06506.

© 1980 AFIPS 0164-1239/80/020169–185$01.00/0

[1] Bowden's first chapter, "A Brief History of Computation," includes a short life of Babbage (pp. 7–22).

[2] See, for example, Bernstein (1964), Rosenberg (1969), and Halacy (1970).